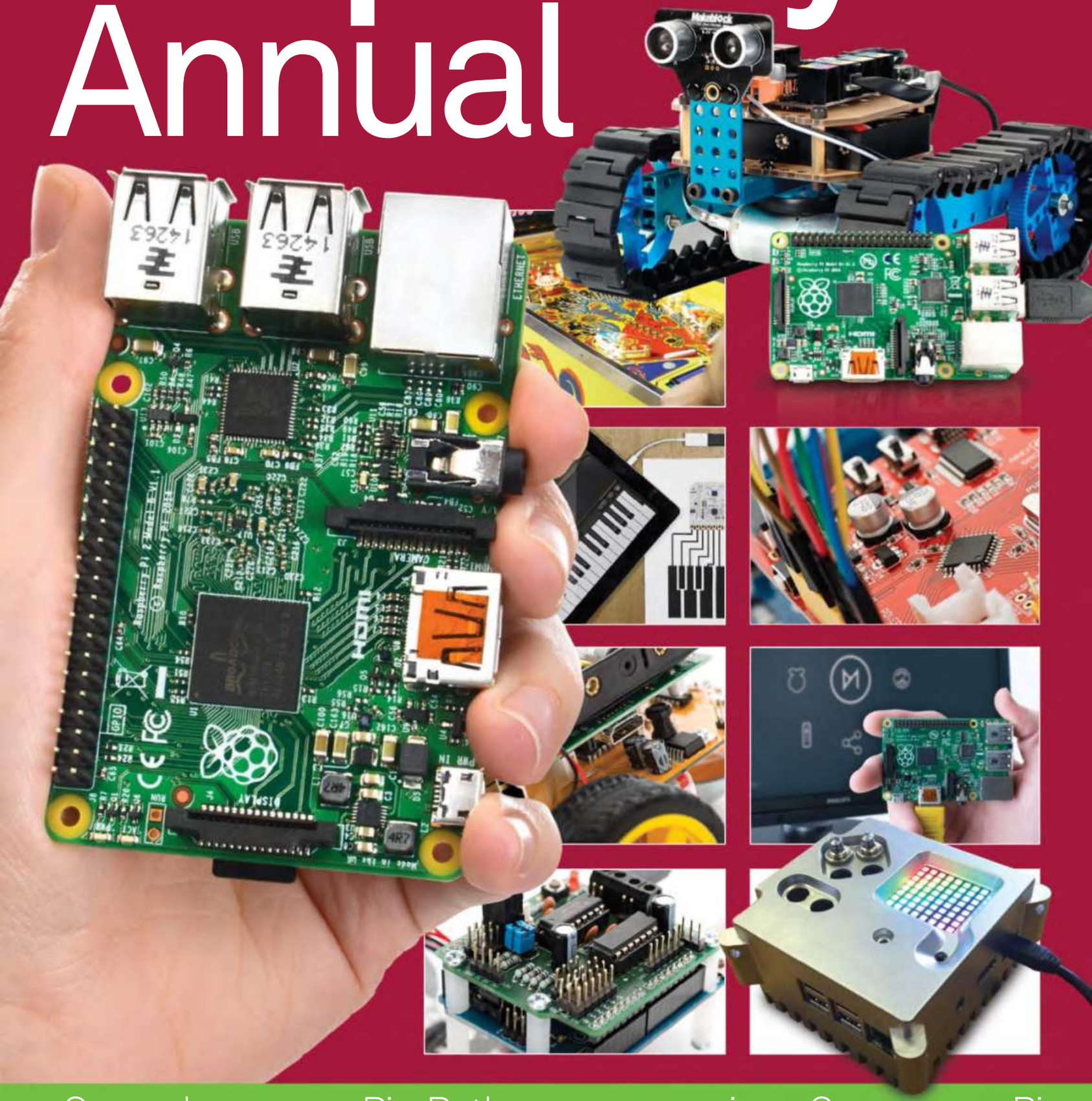
NEW

Everything you need to get the most from Raspberry Pi

100% INDEPENDENT

Raspberry Pi Annual



• Supercharge your Pi • Python programming • Secure your Pi

Welcome to

Raspberry Pi Annual

Since the first Raspberry Pi was released into the wild in 2012, excitement for this powerful mini-PC has continued to grow. Adults and children are being introduced to the world of programming and letting their imagination soar. Builders, coders and hackers from every country are using this amazing device to empower their projects – everything from cloud-seeding drones to self-navigating robots with the senses of sight, sound and touch, not to mention artificial intelligence that the power of Python can bring to the table. We invite you to join the Raspberry Pi revolution. Our second Raspberry Pi Annual comes out as Raspberry Pi gears up to celebrate it's fourth birthday and the staggering 5 million units that have been sold so far. In this book we'll show you how to set up, supercharge and hack your Pi. We'll demonstrate how to start programming with Python. If you've ever wanted to build your own Raspberry Pi robots, we have all the tutorial files you'll need. Let's get started!

Raspberry Pi Annual

Imagine Publishing Ltd
Richmond House
33 Richmond Hill
Bournemouth
Dorset BH2 6EZ
☎ +44 (0) 1202 586200
Website: www.imagine-publishing.co.uk
Twitter: @Books_Imagine
Facebook: www.facebook.com/ImagineBookazines

Publishing Director
Aaron Asadi

Head of Design
Ross Andrews

Production Editor
Hannah Westlake

Senior Art Editor
Greg Whitaker

Designer
Perry Wardell-Wicks

Photographer
James Sheppard

Printed by
William Gibbons, 26 Planetary Road, Willenhall, West Midlands, WV13 3XT

Distributed in the UK, Eire & the Rest of the World by
Marketforce, 5 Churchill Place, Canary Wharf, London, E14 5HU
Tel 0203 787 9060 www.marketforce.co.uk

Distributed in Australia by
Network Services (a division of Bauer Media Group), Level 21 Civic Tower, 66-68 Goulburn Street, Sydney, New South Wales 2000, Australia, Tel +61 2 8667 5288

Disclaimer
The publisher cannot accept responsibility for any unsolicited material lost or damaged in the post. All text and layout is the copyright of Imagine Publishing Ltd. Nothing in this bookazine may be reproduced in whole or part without the written permission of the publisher. All copyrights are recognised and used specifically for the purpose of criticism and review. Although the bookazine has endeavoured to ensure all information is correct at time of print, prices and availability may change. This bookazine is fully independent and not affiliated in any way with the companies mentioned herein.

Raspberry Pi is a trademark of The Raspberry Pi Foundation

Raspberry Pi Annual Volume 2 © 2015 Imagine Publishing Ltd

CONTENTS

10

14

23

25

8 10 Awesome Raspberry Pi upgrades

09 *Fully protect your Pi*
Give your Pi a shell

09 *Portable and solar power*
Take your projects off-grid

10 *Power switch & file safe shutdown*
Shutdown your Pi

10 *See in the dark with infra red*
Nighttime viewing with your Pi

11 *Movement for your camera rig*
Move your rig around

11 *High quality audio*
Get serious about audio quality

12 *High definition display*
Take your screen further

12 *Super low power displays*
Take advantage of eInk diplays

13 *Gesture & touch control*
Add near-field 3D gesture

13 *Control your plug sockets*
Start automating your home

14 Astro Pi
Send code to space

22 20 Raspberry Pi projects

23 *Portable Pi arcade*
Build a hand-held console

24 *Camera Pi*
Power up a DSLR camera

24 *Car computer*
Embed your display in the dashboard

24 *Pi telephone*
Revive a ringing phone with C#

25 *RetroNES*
Resurrect a NES with your Pi

26 *Mission control desk*
Run launches from home

26 *Spaceship bedroom*
A homemade spaceship

27 *Automatic roller*
Get your Pi to wake you up

27 *RasPi Terminal*
Have your heard of Apple Pi?

28 *Alarm clock robot*
Chase your alarm clock

28 *PiFM radio*
Turn your Pi into an FM radio

28 *Ras Pi smart TV*
Make your smart TV smarter

29 *Astrogun*
Shoot down virtual asteroids

29 *Pirate TV*
Totally rebuild Android TV

29 *Pye Radio*
Modify an old radio

30 *Digital camera conversion*
Upgrade an old-school camera

31 *Fireball pinball*
Enter the high score boards

32 *Bitcoin Pool table*
Insert Bitcoins to play

32 *Voice controlled coffee table*
A lightshow coffee table that listens

33 *Project Jarvis*
Build a home automation system

"To maximise your Raspberry Pi, you need to use profiling to figure out exactly where the problems may be"

"In its short life so far, the RasPi has been an absolute game changer"

34 Pi Glove 2
A wearable social media controlling glove

36 Build a Pi Glove - part 1
Build the glove

40 Build a Pi glove - part 2
Create the software for your glove

44 Pi glass
Hack video goggles

46 Visualise music with LEDs
Get the party started

52 Electrosuper
A supersized LED installation

54 Code a simple synth
Write a simple polyphonic synthesiser

60 Build a radio transmitter
Make your mark on the airwaves

64 Stream media
Set up a Samba server and map a network drive to OSMC

68 Set up a wireless speaker
Create a wireless stereo system

70 Fireworks controller
Light up the Fourth of July

72 Forecast the weather
With Python and a Raspberry Pi you can keep an eye on the weather

74 Send SMS from your Pi
Send an SMS from your Raspberry Pi to a mobile phone

76 Working with RSS feeds
Learn how to build a feed ticker

78 Hack your TV with Pi
Build a remote control for your TV

82 Make a visual novel
Bridge the gap between books and videogames with Python and Pygame

86 Stop motion studio
Build your own animation studio by using your Pi as a stop-motion camera

90 Access Twitter with Python
Enable you Pi to talk to the world

92 LEGO smart home
Connect Internet of Things as easily as block of LEGO

94 Study science with a Sensly HAT
Conduct experiments with your Pi

96 Build a Minecraft console
Create a fully functional console

102 Minecraft NFC
Make NFC enabled papercraft

104 RasPiViv
Build an environmental control system

106 RasPi car computer
Make your own touchscreen navigation system

114 Harness the 1-Wire bus
Simplify accessing ready-made sensors

116 Print wirelessly
Use your Pi as a wireless print server

118 Host your own website
Configure your Raspberry Pi to act as a web server

120 Code with FUZE BASIC
Start learning how to code with FUZE BASIC

124 Profiling Python Code
Figure out where the problems are

126 Optimise Python code
Optimise the relevant parts

128 Monitoring the network
See what's happening on your network activity

130 Manage your Pi cluster
Learn how to configure a Pi cluster

132 Optimise by going outside
Use external compiled code

134 Paint conductive circuits
Bring together art and electronics in a whole new way

136 Secure your Pi
Protect yourself with passwords, firewalls and physical security

140 Remotely control your Pi
Use a web interface to control yor Pi and employ it as a fileserver

142 Supercharge your Pi
Get the most out of your Pi with these performance-enhancing tricks

146 Monitor CPU temp
Turn your Raspberry Pi into and Internet of Things

150 What is RasPi robot?
Is the Pi robot a specific product or just a concept?

152 **Our top Raspberry Pi robots**

154 *Rover 5 Seeeduino*
The Seeeduino is fully kitted out and makes a great gift

156 *Pi2Go Lite*
One of the smallest robots tested but has a few tricks

158 *Hexy the Hexapod*
Meet the Kickstarter success story with six legs and mad moves

160 *Frindo*
The punk robot with a low profile and front-facing speakers

162 *Rapiro*
The bipedal, humanoid Arduino and Pi-powered robot

164 *GoPiGo*
A simple and straightforward Pi project robot with WASD control

165 *Scores explained*
We break down our verdicts on these robots' capabilities

166 Remotely control Rapiro
Take Rapiro for a spin without being leashed to a laptop

170 Add web control to Rapiro
Make Rapiro more wireless by installing a remote web interface

10 AWESOME RASPBERRY PI UPGRADES

From solar power packs and ePaper displays to near-field 3D gesture control, here are ten unmissable add-ons for your Pi

In its short life so far of just over three years, the Raspberry Pi has been an absolute game changer – not just as a piece of reduced price hardware, but for nurturing a community of like-minded individuals with a common goal: learning and making awesome stuff!

We can't recall the number of times we've browsed over to the Raspberry Pi blog and been blown away by the brilliance of a new project. From sorting Portuguese mail to making bullet time rigs, there are a lot of incredible projects out there – more and more are surfacing every day. People often ask what they can do with a Raspberry Pi and it is actually sometimes difficult to articulate an answer to that question, as the use cases are so broad that it is hard to do the Raspberry Pi justice.

When comparing the Raspberry Pi to your average off-the-shelf computer or mobile device, the brilliance of the Raspberry Pi comes down to its upgradeability and the amount of customisation that is possible. With a smartphone or tablet you can get a trendy case or some cool headphones to go with it, but the customisation with the Raspberry Pi goes far further than that – both in software and hardware. A lot of projects you look at appear to actually be the real-life manifestations of a childhood dream. That ability to turn what used to be dreams into reality is what makes the Raspberry Pi so well loved.

Here we take a look at ten of our favourite Raspberry Pi upgrades, which will help you bring your ideas to life and serve as some inspiration for your next project!

Fully protect your Pi

Short Crust Plus

£8.99 / $15.95 Available from: **bit.ly/1ICXbvw**

The Raspberry Pi is a durable and reliable little computer, especially when you consider that it is just a populated circuit board with no real protection. However, there may be times where you want to give your Pi a nice shell. Maybe because you want your Pi-based home theatre to look more sleek next to all of your other electronics, or maybe you just want to keep the dust off your tiny computer when carrying it around in your pocket.

The Short Crust Plus is one of our favourite cases for the Model B+ and 2B Raspberry Pis due to its sleek, tidy design and well thought-out features. It is also easy to use – the Pi itself snaps into place inside the case and the lid also clicks into place. Each case comes with a set of self-adhesive rubber feet and a free extension that enables you to increase the height of the case in order to accept any add-on boards you might be using.

Portable & solar power

PiJuice

£25 / $39 Available from: **bit.ly/1Fb1ywy**

You can now get hold of an elegant little add-on board that lets you take your projects off-grid and away from mains power sources. PiJuice is compliant with the Raspberry Pi HAT (Hardware Attached on Top) specification and makes use of a slim, off-the-shelf mobile phone battery, and some intelligent charging and power circuitry, to make your Pi truly portable. There's also a version called PiJuice Solar that enables solar recharging and is even capable of taking inputs from other renewable energy sources.

PiJuice also has a powerful ARM Cortex M0 processor that provides deep sleep functionality, a real time clock, watchdog timers and plenty of other very useful features. The firmware and GUI (graphical user interface) that comes with the PiJuice communicate with the common ACPI (Advanced Configuration and Power Interface) battery and power APIs for tight integration with Raspbian. PiJuice only uses a I2C power sand one GPIO pin, so most of the GPIO pin bank is left free for use with other projects. It comes as standard with a stacking header to make it extremely simple to add other HATs or add-on boards on top. PiJuice will enable you to make a variety of awesome projects – check out the PiJuice Instructables page: **bit.ly/1e2CoGE**.

Power switch & file-safe shutdown

Pi Supply Switch

£15 / $23.10 Available from: **bit.ly/1RXHROn**

The Raspberry Pi has been so popular, in part, because of the extremely good value for money of the hardware. It packs a lot of punch for the price point and, because it is designed by a charity, they don't need to inflate the price with high profit margins as much as would be done with a more commercial product. Unfortunately, as with anything low-cost, some compromises had to be made in order to bring it in at such an affordable and small form factor.

When comparing it to your more standard desktop or laptop computer, one thing that it is obviously lacking is a power switch and power management functionality. It is surprising how something as simple as a power switch can be so very useful, and it is not until you do not have one that you realise this!

The Pi Supply Switch is a self-solder kit which provides an on, off and soft-off (file-safe shutdown) button to give you basic power management functionality for your Pi. With some provided sample scripts you can make sure your Pi is correctly shut down when you switch off – without the need to open any menus or issue any commands in the terminal – and the circuitry in the switch ensures that power is only removed after the Pi has been shut down. As well as making it more convenient for you, it also reduces the possibility of corruption to your SD card from prematurely pulling the power cable.

See in the dark with infrared

NoIR Infrared Camera

£16.80 / $29.95 Available from: **bit.ly/IQyeC4**

The CSI connector on the Raspberry Pi (between the 3.5 mm jack plug and HDMI connector on the most recent models) enables you to connect a camera module directly without the need for a USB-powered webcam. The camera modules that you can connect here use less power and, as you would expect from the Raspberry Pi Foundation, they come in an impressively small form factor – 25 x 24 x 9 mm, weighing in at around three grams (not including the cable).

As you would expect, there is a 'normal' camera module on offer (and by normal, we mean one that captures visible light) with some impressive stats – a 5 MP fixed focus camera, which supports 1080p30, 720p60 and VGA90 video modes (full specs here: **bit.ly/1Gy3D8q**). When the camera module was first released, some people clearly were not happy with a visible light camera and had some other (very cool) applications in mind – so they took apart the tiny camera sensor and removed the infrared filter before putting it all back together again. Painstaking work which obviously voids the warranty, but so many people were doing it that the Raspberry Pi Foundation took notice and started doing it themselves, eventually releasing a new infrared camera module – Pi NoIR.

There are some fairly commonplace nighttime uses for infrared video, and if you pair your Pi NoIR with some infrared LEDs (check out the Bright Pi add-on board for this), then you can easily use it for a night vision security camera or a nocturnal animal monitoring setup. Perhaps most amazingly, if you use the infrared camera in the daytime, it can actually be used to monitor the health of green plants (**bit.ly/1QnZdFG**).

High quality audio for your Pi

HiFiBerry DAC+

£30 / $34.90 Available from: **bit.ly/1L1hh4T**

As an educational tool, the Raspberry Pi is pretty much unparalleled due to the support of the very large community that surrounds it. As a high quality audio device, however, you may think it is lacking due to the fact it only has a 3.5mm stereo output that isn't tuned for high fidelity.

Due to its low cost, small footprint and its ability to act as a home media centre, music and video streaming server and much more, you have probably dreamed of enhancing the audio and taking your setup to the next level. The good news is that the clever folk at the Raspberry Pi Foundation, from the second revision of the original Model B, have provided access to the I2S pins; initially on the separate P5 header, and now on the A+, B+ and 2B models it is available from the main 40-pin GPIO header.

I2S is a communications protocol designed specifically for audio devices and has enabled a number of companies like HiFiBerry and IQaudIO to create high quality audio add-ons for the Raspberry Pi. The HiFiBerry DAC+, for example, is an add-on which brings a high resolution (192 kHz, 24-bit) Burr-Brown digital-to-analogue converter to your Pi. It has hardware volume control using Alsamixer, among other features, and as it is a HAT-compatible board. It works plug-and-play out of the box with the latest Raspberry Pi firmwares, and it works with all the popular operating systems for both standard use and media playback, such as Raspbian, Arch Linux, OSMC, OpenELEC, Volumio, Pi MusicBox and many more. If you are serious about your audio quality and want a high quality, low cost, Internet-connected solution, then you no longer have any excuse – you can build your own for under £100!

Movement for your camera rig

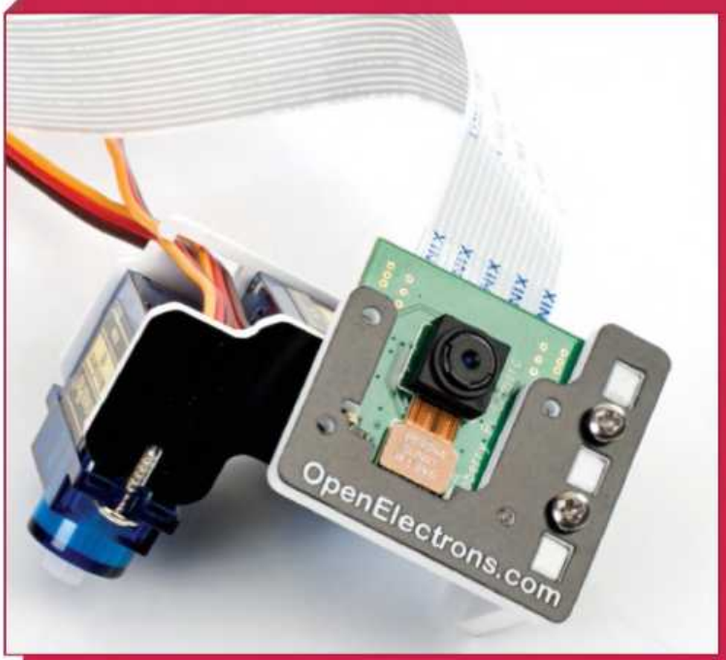
Pi-Pan Pan Tilt Mechanism

£45.99 / $39.99 Available from: **bit.ly/1dwpEr2**

The camera module and Pi NoIR we look at on the opposite page are some pretty essential upgrades to have in your Raspberry Pi toolbox, but what happens if you want to move the camera around to get a different viewpoint? This would be useful in a multitude of projects, such as a robot with a movable camera or an Internet-connected webcam that you can control via a web interface (many IP cameras used for security applications already have a pan-tilt feature, in fact).

The Pi-Pan from Open Electrons is a pan-tilt mechanism for the Raspberry Pi that enables you to articulate the camera by an impressive amount – 110 degrees from top to bottom and 180 degrees from left to right. The kit includes a well considered array of hardware, including a servo driver board, the servo motors required for the actuation and mounting hardware for the camera and servos. On the software side, there are libraries in Python and Scratch so it is easily flexible enough for most projects.

One of the most impressive applications you could use this for is an OpenCV-based motion detection and face-tracking camera. There is sample code available on the **openelectrons.com** forum and it looks like a truly great project to try (**bit.ly/1JJpXLe**).



High definition display & audio

Adafruit 10.1" Display & Audio

£110 / $154.95 Available from: **bit.ly/1HrfR1s**

Finding the right display for your project can often be a bit of a pain. We have covered the HDMIPi in a previous issue (146; **bit.ly/1Gb9LNs**), which is a fantastic 9-inch HD screen for your Raspberry Pi, and it really was wildly successful on Kickstarter (**kck.st/1Culjwd**).

If you want to take things one step further, Adafruit have a 10.1-inch offering that just can't be missed. It features a beautiful 1280 x 800 (so slightly higher than 720p) resolution IPS display with a very wide viewing angle. It has mounting tabs to enable you to easily flush-mount it within your project and it can accept a number of different input methods – HDMI, VGA and composite. Perhaps best of all, this display kit also enables you to directly connect 2-, 4- or 8-Ohm speakers without the need for a separate amplifier or externally powered speaker, which is very useful.

It is not the cheapest display around at $155 on the Adafruit site, but if you need a high quality display in your project with native audio capability then you should seriously consider it. We are already daydreaming of a dedicated multiplayer arcade emulator with built-in stereo audio, and we're sure you can come up with some cool applications too!

Super low-power displays

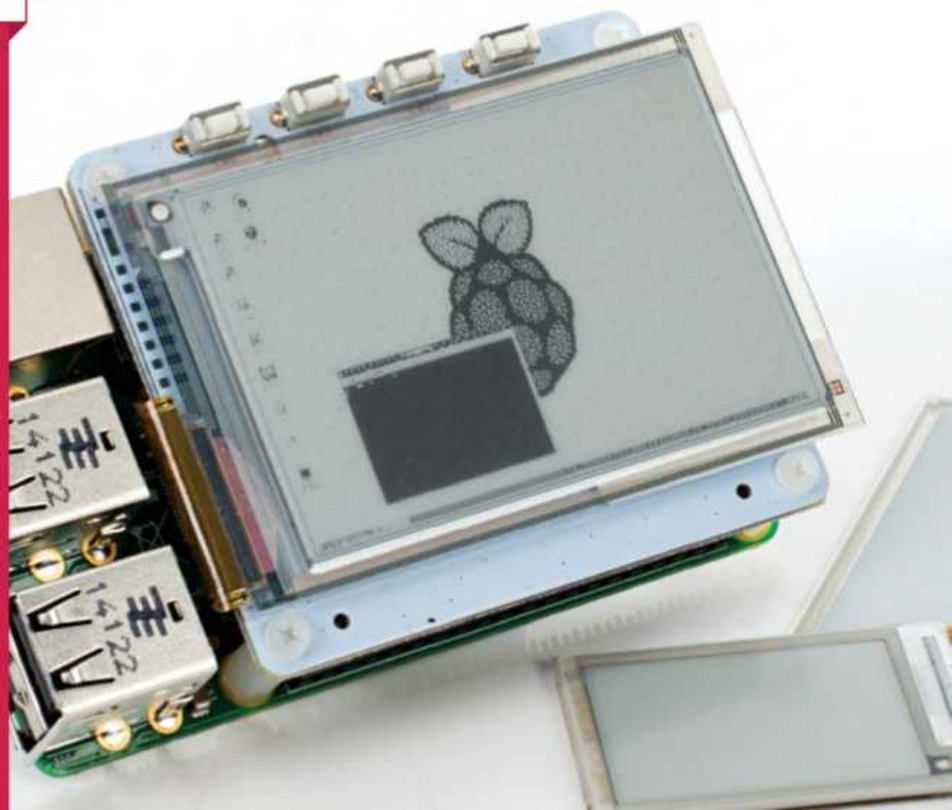
PaPiRus ePaper/eInk HAT

£30-65 / $47-102 Available from: **bit.ly/1f2Lzaj**

As computers of all sizes and powers are now being embedded into pretty much everything, electronic parts have become even more commoditised and, happily, this is filtering down to display technology as well. We now have a wealth of offerings from your standard monochrome LCDs to TFT, OLED and AMOLED offerings.

One of the most exciting and disruptive display technologies of recent times is ePaper/eInk. You probably know it best as the screens that go into e-readers like the Kindle and Kobo (fun fact: the Pebble watch is commonly referred to as an ePaper watch, but it actually uses what is known as a Memory LCD and a very clever marketing team). You may have wondered in the past why your iPad barely lasts five hours on a charge but your Kindle lasts for over a week, and the answer is all to do with the display. ePaper only uses power to update what is on the screen, which means that for a large number of applications where you don't need to change screen contents particularly often, it saves a lot of battery power. It would be pretty useless for playing videos, but for e-readers, monochrome graphical info displays, digital price tags, bus and train station signage and many more applications, it is by far the best choice.

PaPiRus brings the low power ePaper display technology you know and love to the Raspberry Pi in a HAT-compatible format with screen sizes ranging from 1.44 to 2.7-inches. The ePaper film used in these screens is actually identical to that in the popular e-readers mentioned above. You can get your hands on one for around £35 and they come with a useful Python and command line framework. They are worth trying out if you have any display-driven projects!



Gesture & touch control

Pimoroni Skywriter HAT

£16 / $20.95 Available from: **bit.ly/1lFt9cg**

For a lot of projects you undertake with the Raspberry Pi, you will want some kind of user interaction. When using the desktop GUI this is normally done with a keyboard and mouse, but these are not always the most intuitive input methods when you aren't using a full desktop environment and when you don't need to type anything.

The pirates over at Pimoroni have created a new HAT module called the Skywriter that enables you to add near-field 3D gesture and touch sensing to your projects for a great price. There is a Python API provided that provides full 3D position data and gesture information (swipes, taps and so on). Play with this for a short while and you will realise that it is a really nice input method with a lot of potential – Pimoroni even have a video of a home-made Ras Pi-based theremin (**vine.co/v/OrUWTdd0Hlg**).

There is even a larger non-HAT version of the Skywriter that is more than twice the size and boasts a sensing distance of around 15 cm, which means that you can mount it inside your projects behind a sheet of acrylic or other non-conductive material and it will still work. This is especially good if you want to convince people that your projects are simply pure magic.

Control your plug sockets

Energenie Pi-mote Control Starter Kit

£19.99 / $31 Available from: **bit.ly/1L1kYHU**

Home automation is all the rage at the moment – perhaps it is because people are inherently lazy or maybe it's just because this tech is extremely fun to play with! Either way it doesn't really matter, as it can make our lives easier and quicker and can automate tasks that would often be boring and monotonous, like fiddling with heating controls and turning off the lights before you go to bed.

One thing that we are always told is to turn off devices at the plug rather than leaving them on standby, as they use a lot of electricity when not properly turned off. This is sound advice but is not always a practical solution as the socket is not easily accessible. This is where the Energenie Pi-mote control starter kit comes in. It contains two remote-controlled plug sockets which can be turned on and off with an RF remote. What does this have to do with the Raspberry Pi? Well you also get an add-on board to enable you to control the sockets via software on the Raspberry Pi, which unleashes whole new possibilities – you could set your lamps to turn on and off automatically at specified times when you are away to avoid burglars, or create a basic web app to control your plug sockets remotely using your smartphone.

They only come in UK and EU plug types, so if you use a different plug then you may need to look for something else (and maybe send Energenie a request to make more versions).

Astro Pi: Sending code to space

Clever Year 7 students at Thirsk School have devised an amazing tracking system for the International Space Station and have become Astro Pi competition winners. We speak to their teacher, Dan Aldred, to find out more...

What is the Astro Pi competition?

Run jointly by the Raspberry Pi Foundation and leading UKspace companies, the competition set primary and secondary school students the challenge of creating an innovative project using the Raspberry Pi and a specially designed Astro Pi HAT module, which is packed with sensors and a colourful LED display matrix. The winning teams have now been announced and all their code will be sent up to the ISS in December, along with astronaut Tim Peake and a bunch of Astro Pis, where the greatest school experiments off Earth will begin.

Can you tell us more about your students at Thirsk School who won the competition?

It was actually a code club that I'd set up at lunchtimes. The original reason for setting it up was to give students, who were perhaps what we call vulnerable learners, something to do at lunchtime – students who would struggle being in the playground; maybe their behaviour means they would get into difficulty, or they were just a bit more timid and so didn't have anywhere to go. Also, I was keen on making sure that the coding and the Raspberry Pi wasn't about bright kids – I wanted to make sure that low-ability kids and special needs kids had access to coding and all the benefits that it offers.

So I set up a coding club for lunchtimes, started with *Minecraft*, Sonic Pi, picamera photo hacking, and then this competition came along and I said, "Look, this is the opportunity we've got: a space rocket's going to go up to the ISS with an astronaut and an Astro Pi. What do you think?" They were like, "Yeah! Let's do it, let's do it!" And it grew from there – we ended up with eight to ten students who stayed every lunchtime for seven weeks, creating their winning solution.

That's amazing dedication!

It is! In the end it became quite social, and by about week four they could see the results of what they'd made and start to get excited, thinking that it could actually win. But yeah, the dedication from them was huge, really motivated.

It must have been great for building a sense of community too, particularly with the vulnerable learners.

It was very exciting and rewarding personally, too. We started off with a shared document, so all the students could access the code from home, and what I found was that as the weeks went on, the students were logging in more frequently to check their code, add their sections, and then they started editing each other's code. It was so polite – they'd come in at lunchtimes, for example, saying, "I noticed an error in your code last night. I changed it – hope you don't mind?" And then of course they had a common goal they could talk about, and they started talking about space and physics, different space films they'd seen, and of course as we were creating it they were talking about the different countries, whether they'd been to that country, what it's like, what's the capital – at work we call them learning conversations; they were learning just through talking around the subject.

Organic, peer-to-peer exchange.

Exactly – it wasn't manufactured. It was completely natural, which was absolutely brilliant. But yeah, they've forged some quite good friendships. And confidence as well – these are students who perhaps at the beginning when they started school (they were Year 7 students, so they're 11 years old now) wouldn't really go into the playground,

Above The ISS will use the winning tracking system to predict which country it is above

Right Here are some of the Space-Byrds, the team behind the awesome ISS tracker

were perhaps fearful of the dining hall, were maybe bottom-set students struggling with maths and English, or had a behaviour issue, and suddenly they've got quite a good status now amongst Year 7. And obviously the press have gotten hold of it and various local papers have run pieces on it and printed pictures of them, and I think it's given them a real boost. Rather than being labelled as an underachiever, a pupil premium child, having the potential to achieve, etc – well, they have all actually overachieved now!

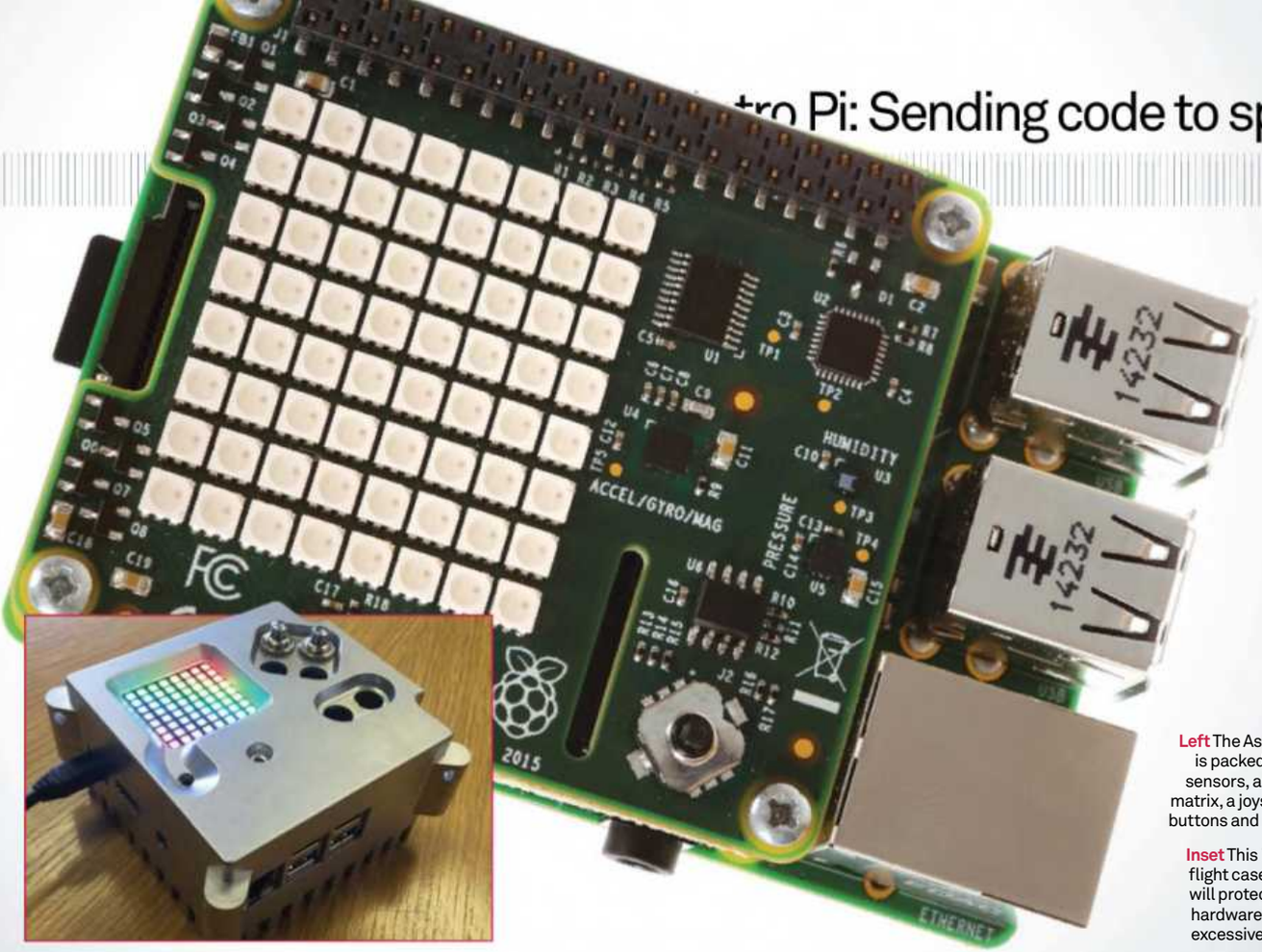
It must have been amazing for their confidence and social skills, working in a collaborative environment like that.

Definitely. The program that they made was actually very simple in essence – it's just the fact that it's so big that took the time. In terms of the coding, it wasn't particularly difficult; it was just a case of there being 96 countries filled with 96 different flags, and 96 different languages that you have to find out and create messages for. So once they'd mastered the skill they were learning, it was then a case of repetition and discovery. The bit that was individual at that point was that the flag for Kazakhstan is different to the flag for the UK, and stuff like that. But creating each flag is a similar set of code – obviously the colours are slightly different, and the setup, but in essence the code was the same, so they could support each other, and say, "Oh, actually you've missed this bit out on the red, green and blue – you haven't got a value for the blue, that's why it's this colour." So yeah, they've learned a heck of a lot of skills and they've also learned a lot about the other countries as well, through supporting each other.

In terms of the logistics, how did the division of the work happen at the beginning and the end of the project?

There were two parts to the competition: the first was to pitch an idea, and you were then selected from that to go into the second stage. So the first couple of lunchtimes it was basically just brainstorming ideas, listening to what everybody wanted to come up with. We had some fantastic concepts, like, "Can we strap it to the astronaut, so that when he or she goes outside the ISS it can check for radiation?" Despite having the great ideas, we didn't quite know how much of it was realistic! I contacted Raspberry Pi and asked for a breakdown of what we can and can't do, and when we got the breakdown it said it was going to be stationary, it was going to be inside the station, it's not going to be moving, there's going to be no screen and the astronauts really need to have minimal interaction with it, other than maybe starting it up and pressing a couple of buttons. So then we could shrink down the list, and I suppose the final idea came out because one student said, "So they're in space... how do they know where they are?" We talked about the different instruments and the fact they've got GPS or an equivalent tracking and co-ordinating system, but when they look over a country, how do they know which one they're looking over? And that's where the idea came out – why don't we

Below This is Tim Peake, the British ESA astronaut who'll be taking the projects into orbit



Left The Astro Pi is packed with sensors, an 8x8 matrix, a joystick, buttons and more

Inset This is the flight case that will protect the hardware from excessive heat

have our Astro Pi system show the astronauts the flag of the country and a message, so they could compare that with the instruments on-board the space station and see if it works? So they all decided on that, we pitched it to Raspberry Pi, who said it was a great idea and sent us the kit, we got started, and picked out 96 major countries. For that, the students used the ISS trackers online and basically looked at the plot map of where it goes. It was quite a time-consuming process because they had to write down all the countries they were going to complete and put them into a shared Word document. I then put the example code at the top for England with the UK flag – from there they just had to work up the countries. Towards the end of the project we had a couple of students who'd set up a spreadsheet with all the 96 countries, 96 flags, 96 messages, and they began ticking them off.

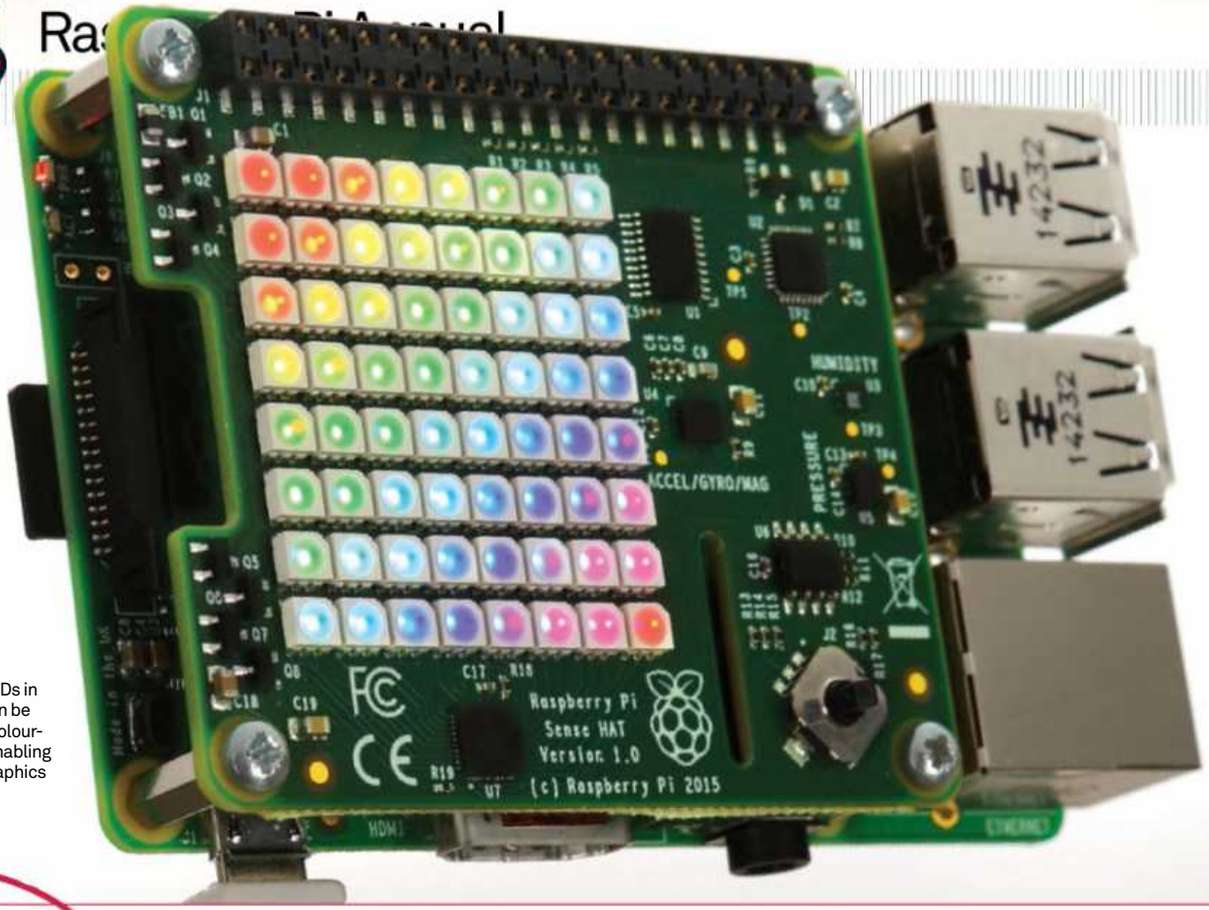
And we had a couple of Astro Pis – one to test the flags and then the other was running all the co-ordinate tracking, so some of the students began working on that. It was probably by week five that we started to integrate the two together, so that if the ISS positional data was within the boundaries of the country then the flag pops up. Towards the end we could start to refine the longitude and latitude so that you got an exact position for the country. One student was in charge of finding out all the longitudes and latitudes for the countries – an absolutely painstaking job because there were four points of origin for most countries, and there are some countries in L shapes so we had to do six or eight points. It's not perfect – it's quite a crude model and we're looking at a way of making it more accurate – but for the purpose of saying we're over Australia, for example, if you're within these four points of longitude and latitude then you're within the boundary. So one student was responsible for that.

So where exactly is the Raspberry Pi getting all of the longitude and latitude data from?

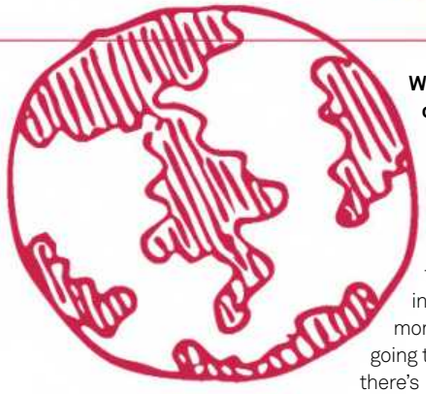
> It works out which country's territory the ISS is above and shows its flag on the LED matrix along with a short phrase in the local language

Here's the official press release of it: "the program uses telemetry data provided by NORAD along with the real-time clock on the Astro Pi to computationally predict the location of the ISS so it doesn't need to be online. It then works out which country's territory the ISS is above and shows its flag on the LED matrix along with a short phrase in the local language". So that's the official blurb.

The coding bit for the flags etc was tricky, but the mathematically challenging bit was the TLE file, which was a two-line element file that looks at the time on the Raspberry Pi and makes a calculation of where the ISS should be. From that it returns the longitude and latitude position. The students wrote conditional statements – if it's within this longitude and latitude then it must be over this country, and therefore return this flag; and if it's not then it displays a little graphic and says 'calculating current position'. The experiment was comparing that set of results off the Raspberry Pi with what the ISS tracking system actually says on-board. It makes 19 orbits a day and can go out of sync, so the TLE file is updated 19 times a day. You have to download those two lines of code, pop it into your Python program and then it calculates the new positions. One of the biggest challenges was getting the time correct, but the Raspberry Pi Foundation has been great – it worked with us to ensure that it's accurate when the Raspberry Pi boots up, that the Astro Pi and Raspberry Pi are in sync, and that it's the correct time.



Above The LEDs in the matrix can be individually colour-controlled, enabling some cool graphics



What's the next step for the project, then – are you completely ready for launch day, just waiting for Tim Peake to go up?

Yep – Raspberry Pi has been in contact. Tim's going up in December but on the 11th August he's doing a test run in Germany, which basically involves him being in a simulation for a number of weeks, and within that simulation he will run a number of experiments, including our ISS tracker experiment. So the code at the moment, the project we've built, is staying as it is and it's going to be used as a test run so Tim can check it works, that there's no malfunctions, etc. And then in December he will fly up to the ISS and begin experiments there for six months, sending the data back to the UK.

So at that point, will you be running the experiment concurrently with an Astro Pi at Thirsk School?

Yep – as soon as we get confirmation he's on board, we're going to set up a copy of the ISS tracker and record the data from each day, and then with the two pieces of data that he returns – the ISS' real position from their flight instruments and then our ISS Astro Pi tracker – we'll compare all three.

In terms of September when we return to school, the maths department are on board now and they are going to build us a pixelated map of the world, where each element of the country boundary will be within a pixel grid reference, so what we can actually do is take the longitude and latitude of each country and break it down to a pixel position. At the moment, what we've had to do for ease of use for the students is basically draw rectangles or squares around the countries using four points of origin, or with countries like Columbia, which is L-shaped, we've drawn a rectangle at the top and a rectangle at the bottom to get six points. So it's accurate, but with somewhere like Russia and Kazakhstan, as it goes over it actually undulates between the two different countries, so for two minutes it's in Kazakhstan and then for two minutes it goes into Russia and back out again. So for that kind of thing, our measurements weren't accurate enough to show that, but obviously a pixelated version of the atlas is going to be better.

I bet you'll have an awesome live-updating map going once you've got the pixel map sorted!

That's a good idea... I'd also like to set up some kind of live web feed so that everyone can compare the live ISS data with what our live Astro Pi ISS tracker is saying. A lot of the parents have contacted me, saying, "This is great – my son/daughter is talking about this and they're so excited." I'm going to share some pictures on Facebook and Twitter because I think when people actually see it, they'll understand it better. If I put a picture of some LEDs showing the Brazilian flag and say it's tracking the ISS, it doesn't really mean a lot. But if you can see there's the ISS over Brazil, and here's the Astro Pi with the Brazil flag, and now it's going over Columbia you can see the flag change, and oh there's the language...

When it started, the club was just running every Monday – now we're up to every lunchtime, five days a week. And we've got a beginner's club on Monday, so what happens is the students who've been doing it since November last year come along and they support the new kids, and they feel really good now because they know everything – sudo idle and all the different commands – and they remember how they were when they first started. And they don't go to the club saying, "I'm going to learn coding." They go there saying, "I want to build a car that we can control from the computer. I'm going to build a tank. I'm going to play the *Mario* theme tune in Sonic Pi. I'm going to turn the water to ice in *Minecraft* just by walking on it." And that's what inspires them to do it. Exciting, isn't it?

Want to keep reading about this fantastic project? We couldn't fit the whole conversation into this article but you can read the uncut version of this interview online: ***www.linuxuser.co.uk/news/astro-pi-space-byrds***.

Track the International Space Station

Use a Raspberry Pi and Astro Pi HAT to code the winning ISS-tracking program

British ESA astronaut Tim Peake has been preparing and training for his stay on-board the ISS, which begins this December 2015. The ISS orbits at around 400km above our heads at an incredible speed of 17,150 miles per hour, so this means that the crew orbit the Earth every 92 minutes. As part of his six-month mission, he will run a number of Astro Pi experiments created by schools, each judged and selected by the ESA, the National Space Agency and Raspberry Pi.

The Astro Pi HAT has been designed and built specifically for this mission and boasts an array of sensors and an 8x8 LED matrix. Each experiment will generate and collect data which will then be downloaded to Earth for analysis. In this tutorial, you will be introduced to some Astro Pi programs and learn how to create a program to track the longitude and latitude of the ISS in real time. If you do not have an Astro Pi, skip ahead to Step 9.

01 Install the Astro Pi software

Attach the board to the GPIO pins and install the Astro Pi software, downloadable from the hosted Astro Pi website. Boot up your Raspberry Pi, load the LXTerminal and type in the following code on a single line. On completion, reboot your Raspberry Pi, typing **–sudo halt**.

```
wget -O - http://www.raspberrypi.org/files/astro-pi/
astro-pi-install.sh --no-check-certificate | bash
```

02 Example programs

The software comes with a few starter programs that can be used to test that the Astro Pi is functioning correctly and to demonstrate some features of the board. The example programs are stored in the /home/pi/astro-pi-hat/examples folder and run in Python 3 IDLE.

03 Take a temperature reading

The Astro Pi has a built-in thermometer that can be easily coded to read and return the current temperature. The sensor is fairly close to the CPU and may pick up some residual heat, however on the whole the reading is sound. To measure the temperature on the Astro Pi, open your Python 3 editor and type in the code below, then save and run it. The current temperature reading will be returned in the shell.

```
from astro_pi import AstroPi
ap = AstroPi()
temp = ap.get_temperature()
print("Temperature: %s C" % temp)
```

04 Compass reading

One of the nifty sensors on-board is the compass. This can be used to return a measurement of the Astro Pi's position in relation to magnetic north. The code is simple to use: **ap.get_compass()** in line 3 (below) returns the position which is stored in a variable called north. The value that is measured is then printed out in line 4. Use this code example to test the compass sensor and the readings:

```
from astro_pi import AstroPi
ap = AstroPi()
north = ap.get_compass()
print("North: %s" % north)
```

> The Astro Pi HAT has been designed and built specifically for this mission

05 LEDs

The 8x8 LED matrix is programmable and includes a range of colours and brightness settings. Each LED can be coded individually and combined to create a simple image. To set an LED colour, create a variable and assign an RGB value to it. In line 3 (below) the colour is set to red, using the values (255, 0, 0). Add additional colours by creating additional variables and setting the RGB codes for each new colour. Then create a representation of the image using the variable names – in this example, the X and O symbols (line 6) combine to create a question mark. Set the LEDs with the code **ap.set_pixels(question_mark)** in line 7.

```
from astro_pi import AstroPi
ap = AstroPi()
X = [255, 0, 0]  # Red
O = [255, 255, 255]  # White
question_mark = [
O, O, O, X, X, O, O, O,
O, O, X, O, O, X, O, O,
O, O, O, O, O, X, O, O,
O, O, O, O, X, O, O, O,
O, O, O, X, O, O, O, O,
O, O, O, X, O, O, O, O,
O, O, O, O, O, O, O, O,
O, O, O, X, O, O, O, O
]
ap.set_pixels(question_mark)
```

Above Converting your own images is a great way to speed up the creation of LED matrix graphics

06 LED per pixel

The image on the LED matrix can also be set automatically from an image file. For example, an image of a space invader can be loaded, the colours and positions calculated and then the corresponding LEDs enabled. Ensure that your image is 8x8 pixels in size and save it into the same folder that the program is saved within. Use the code below to open and load the image of the space invader image – the Astro Pi will do the rest of the work:

```
from astro_pi import AstroPi
ap = AstroPi()
ap.load_image("space_invader.png")
```

07 A single letter

The LED matrix can be used to display a single letter using the simple code line **ap.show_letter(str(a))** – this code would display the lowercase letter 'a' on the matrix. Using a for loop and a range function (line 4), you can create a simple countdown that displays numbers from 9 to 0. Note that the list is reversed; this enables the numbers to count down from 9 to 0.

```
import time
from astro_pi import AstroPi
ap = AstroPi()
for i in reversed(range(0,10)):
    ap.show_letter(str(i))
    time.sleep(1)
```

08 Scroll a message

Writing code to scroll text on LCD/LED displays can be challenging and frustrating. The Astro Pi API removes the difficulties and simplifies the whole procedure to a simple line of code: **ap.show_message("This is a test message")**. Change the text between the quotation marks, save and run the program, and your message will then be scrolled across the Astro Pi LEDs. Adjust the colour of the message and the time it takes to scroll by including **text_colour=[255, 0, 0]**, setting an RGB value, and **scroll_speed=(0.05)** within the function's brackets. Try experimenting with this example code:

```
from astro_pi import AstroPi
ap = AstroPi()
ap.show_message("Linux User and Developer", text_
colour=[255, 0, 0])
```

Pixel perfect

The Astro Pi's LED matrix is 8x8 in size and there are several websites and apps that can be used to mock up an image, modify and create a suitably sized image, for example: **gurgleapps.com/tools/matrix** or **piq.codeus.net/draw**

09 Install PyEphem

The remaining steps cover the program to track the ISS in real time. PyEphem provides astronomical computations for the Python programming language. Given a date and location on the Earth's surface, it can compute the positions of satellites whose orbital elements the user provides. The ISS is technically a satellite as it orbits the Earth, therefore the PyEphem library can be used to track it. Install the software using Pip:

```
sudo apt-get update
sudo apt-get upgrade
pip install pyephem
```

10 Import the required modules

For this and the following steps, refer to the annotations in the full code listing on the opposite page. Open a new window in IDLE 3 and import the modules shown. These import the Astro Pi API, the position tracking program and the time functions to allow you to add pauses or rests to your program.

11 The TLE file

To calculate the position of the ISS you will need to use an up-to-date Two Line Element (TLE) file. The TLE is a standard mathematical model to describe a satellite's orbit and is processed by tracking software. The data results returned include predictions for viewing times, speed and the current position, which is returned as longitude and latitude values. The TLE data is available from the NORAD website and is updated several times a day: **https://celestrak.com/NORAD/elements/stations.txt**. Go to the site and copy the first three lines of data at the top of the page.

12 Prepare the TLE data

Before you can use the TLE data, you need to ensure that it is set up correctly – if it isn't then you will receive errors. In your Python program, create three new variables called **name**, **line1** and **line2**. Next to the name variable add the name of the satellite: ISS (ZARYA). Now add the data from line one of the TLE to the variable called line1. Do the same for line 2; adding the second line of data. Ensure that the layout of the variables remains the same, as shown in the full code listing.

The data shown here is an example; the current data values and their formatting can be found at the NORAD site: **https://celestrak.com/NORAD/elements/stations.txt**

13 Calculate the position of the ISS

The TLE data is now ready to calculate and predict the position of the ISS. A further three lines of code will enable you to retrieve the data. The first line, **tle_rec = ephem.readtle(name, line1, line2)**, creates a variable of TLE data. In line two, **tle_rec.**

compute(), the maths is crunched and the position calculation is performed. Once this is completed, extract the required longitude and latitude measurement data using the line **print tle_rec.sublong, tle_rec.sublat**. You can compare the result with an online tracker such as **isstracker.com**. Remember that the accuracy of the TLE prediction is based on the clock time of your Raspberry Pi – ensure that this is accurately set.

14 Convert to a string and split

The data retuned is very accurate and you will note that the **tle_rec.sublong**, **tle_rec.sublat** data can be up to nine decimal places in length. You may find that this is too accurate for your measurements as most countries' longitude and latitude are given to a single decimal place. In order to reduce the decimal places, you need to convert the data to a string and split the data at the colon. Create two new variables and use **str** to convert the data to a string, as shown in the first two lines. Use **split(":")** to split and return usable data, as shown in the next two lines.

15 Print the data

Once the data is tidy and usable, convert it back into a float number. This is handy for using the values to check the location that the ISS is currently flying over and compare this with a country's boundaries (see Step 17). Convert the variables back into a float value using the code: **lati[0] = float(lati[0])**. In the first two lines, the [0] ensures that only the value is returned with the first decimal position. The next two lines check that the data returned is usable; they aren't needed in the final code.

16 County comparison

Now that you have the longitude and latitude positions for the ISS you can begin to compare these with the positions of cities, capitals and countries, plotting the location. There are many websites that list the positions of a capital city – for example, **csgnetwork.com/llinfotable.html**. You can also use websites such as **itouchmap.com/latlong.html** to plot the boundaries of a country in terms of its longitude and latitude. This is challenging, as some countries undulate between two or three borders. You will find it easier to take a rough approximation of the countries' shapes and co-ordinates.

17 Comparison with position data and county

The final step is to take the data and compare it with the country boundary data – ie if the ISS is within *this* range then it is within *that* particular country's boundary. Create a simple conditional using an if statement to check when the ISS flies over, say, the UK. Use a print statement to display the name of the country. You can also use the LED code from Step 5 to create a flag of the county that is displayed as the ISS flies over the country.

Full code listing

Step 10
```
from astro_pi import AstroPi
import ephem
import datetime
import time
ap = AstroPi()
```

Step 12
```
name = "ISS (ZARYA)";
line1 = "1 25544U 98067A   15185.95963984  .00006354  00000-0
         98170-4 0  9990"
line2 = "2 25544  51.6454 355.2696 0003202 121.3230  14.1346
         15.55509232950800"

ap.clear()

while True:
```

Step 13
```
    tle_rec = ephem.readtle(name, line1, line2)
    tle_rec.compute()
```

Step 14
```
    #convert to strings#
    lat2string = str(tle_rec.sublat)
    long2string = str(tle_rec.sublong)

    #Split to pull out data
    lati = lat2string.split(":")
    longt = long2string.split(":")

    ###Convert to floats to check the rangess
```

Step 15
```
    lati[0] = float(lati[0])
    longt[0] = float(longt[0])
    print lati[0]
    print longt[0]

    ###Check the location###
```

Step 17
```
    ###UK###
    if (lati[0] <= 53 and lati[0]>= 52) and (longt[0] >= -4
                                        and longt[0]<= -1):
        print "United Kingdom"

        X = [255, 0, 0]  # Red
        O = [255, 255, 255]  # White

        UK = [
        O, O, O, X, X, O, O, O,
        O, O, O, X, X, O, O, O,
        O, O, O, X, X, O, O, O,
        X, X, X, X, X, X, X, X,
        X, X, X, X, X, X, X, X,
        O, O, O, X, X, O, O, O,
        O, O, O, X, X, O, O, O,
        O, O, O, X, X, O, O, O
        ]

        ap.set_pixels(UK)
        time.sleep(6)
        ap.show_message("Hello ISS, you are over the UK")

    else:
        ap.show_message("€œChecking location")
```

20 Raspberry Pi Projects

Get the inside story on the greatest Raspberry Pi hardware hacks

There are now over five million Raspberry Pi models out in the wild, and some of the things that you, the Raspberry Pi community, have made with them truly are wild. From elegantly crafted scripts that chain together a series of web services to homebrew Rube Goldberg machines, they are as creative as they are diverse. And through the crowd of new projects bubbling up online every day, if there's one word that's guaranteed to get everyone's attention then it's the word 'hack'.

But what exactly is a hack? Well, for the purposes of this feature, we have decided that a hack has to have some sort of hardware base to it. It's the kind of project where you take one device and, with a little bit of Raspberry Pi magic, transform it into something wholly new and completely original. These are the kind of projects that get us excited and make us want to learn more about electronics, engineering and programming.

Over the next few pages we're going to introduce you to some of the greatest Raspberry Pi hacks that we've ever discovered. Projects where vintage hardware has been torn apart and the components repurposed into something amazing, or where the hardware has been puzzled over, fiddled with, then brought back to life after twenty years spent in a garage. These hacks inspire us, with each maker striking the right balance between passion, skill and virtuosity, and we hope they inspire you too. Read on as we hear how you can launch a satellite from a bedroom spaceship, transform an analogue camera into a digital one, make a classic Apple Pi and more.

Portable Pi Arcade

Left Install MAME and you will never run out of arcade games to play

Love old arcade games? With Ben Heck's hack, you can play them all on a single hand-built, hand-held console

Ben Heck has built two versions of the Portable Pi Arcade. The first was the original Portable Raspberry Pi project (**youtube.com/watch?v=dUZjzQuTNX4**), where he hacked the Pi to reduce its size and opened up a USB game controller to extract the circuits. With a new assembly in place, he 3D-printed a custom-designed case, put the new device together, and booted up MAME (the Multiple Arcade Machine Emulator) to play old-school games.

His recent revival of this earlier project was even more home-made. For the Raspberry Pi MAME portable gaming device (**youtube.com/watch?v=zrEj1aQRbpw**), Ben made a circuit board from scratch, fitting all the components into a new 3D-printed case that, rather than resembling a Game Gear, looks pretty close to a Game Boy.

Above Watch the video to see how Ben build and tests the custom circuit board

We asked Ben to take us through the original version of his project: "My first portable Pi project was a small, battery-powered unit for gaming," he begins. "It had a single USB port for Wi-Fi or external storage and we featured it back on season three of the show. The screen came from a cheap NTSC LCD screen that Amazon sells for use as a car's backup camera. The buttons I laser-cut myself and the case was 3D-printed." And in terms of physical modifications to the Pi? "Mostly I removed the taller through-hole components," he replies, "and attached the Teensy HID (used for controllers) directly to it. I also moved the secondary USB port."

As you can see above, the case is very well made. "I did the initial layout in Adobe Illustrator, for the laser-cut portions," explains Ben, "then transferred the whole design to Autodesk 123D to create a 3D-printable file. Hand-writing the buttons for the controls was the most challenging part of this project. It was the most time-intensive part and required a lot of precision and attention to detail."

Ben is no stranger to taking apart consoles and controllers for his Pi hacks – but he also makes one-handed accessibility controllers. "In addition to all of the other projects and hacks, we modify gaming controllers for people who have difficulty using existing ones," Ben tells us. "On the show we've featured a few of them – Xbox One, PS4, even the Wii. Now we build these controllers by request and they can be ordered off my website, though we only do Xbox 360/Xbox One controllers as those use PCBs throughout (instead of silk screen circuits like the PS4). Recently I trained Felix (an assistant on element14's *The Ben Heck Show*) on how to do it, so he's been helping and working on them in his spare time as well."

Maker Profile

Ben Heck

Master hacker, creator of The Ben Heck Show

Ben Heck is an online sensation and a pillar of the maker community, putting out amazing how-to videos for games console hacks and all kinds of different Pi projects. He's done it all on *The Ben Heck Show*.

Find out more: **benheck.com**

Heck's hacks

Pi Point and Shoot: A Raspberry Pi camera module, PiTFT from Adafruit, PlayStation 3 controller battery and additional parts were all made into a point-and-shoot camera.

Pi Retro Computer: A tribute to the BBC Microcomputer from the 1980s, Ben mounted a Raspberry Pi to a self-made wooden case, HDMI port, on/off switch and USB hub for an 'old-school' feel computer and carrying case.

Handheld Pi Console: Ben hacked a Raspberry Pi single board computer to make it smaller. Combined with a composite LCD wireless keyboard, lithium battery power source and USB joystick, he created his own handheld gaming console.

Camera Pi

Power up a regular point-and-shoot DSLR camera

Above
Read up on the full build process and check out Dave's video at **bit.ly/1BxEMbC**

Maker Profile

Dave Hunt

Photographer and maker

David has been making projects for the Raspberry Pi since the early days.

Find out more:
davidhunt.ie

We last spoke to Dave Hunt back in issue 141. This issue he's back with his Camera Pi.

"I needed to transmit photos to an iPad as they were taken," explains Dave, "but the commercial solutions were £500. I had a broken battery grip big enough to fit my Raspberry Pi and a battery, so it went from there.

"The battery grip holds two batteries. Once I'd stripped out the battery compartment, I set about filing down all the mounting holes inside the grip so I could get the Raspberry Pi inside.

"The next task was to fit a camera battery and DC-DC converter inside. I was able to use part of the removed internals of the grip, and before long I had a slot to insert a camera battery into. It's capable of powering the Pi for about four hours.

"Making it wireless was a case of plugging in a USB Wi-Fi adapter. A few lines of Perl later and I was able to poll the camera with gphoto2, pull the new files off and send them via FTP to ShutterSnitch on my iPad."

Pi Telephone

Revive a ringing phone with C# circuit wizardry and voltage manipulation

Maker Profile

Stuart Johnson

Managing director

Stuart runs Logic Ethos, an IT company in Southampton providing network services and cloud computing to developers.

Find out more:
logicethos.com

Stuart Johnson is bringing a classic GPO 746 handset back to life, and while the project isn't yet complete, he has finished the lion's share of it.

"I took out the main circuit board inside the phone and squeezed the Raspberry Pi in there," says Stuart. "I was then faced with two challenges – the biggest one was getting the bell to ring. I found a solution by raising the voltage to 19 volts and dropping it down to 5 for the Ras Pi using a very small DC-DC converter (the OKI-78SR), with the rest then being used for the bell. I was surprised by how well it worked.

"The bell is using one of the I/O ports, and there's an available C# library (raspberry-sharp-io) which lets you monitor and control those ports. So I linked one of the I/Os to the pulse dial and connected another to a relay using transistors. Then with the software I put in a timer to measure the pulse clicks. I managed to write some code to time those pulse clicks and determine the number dialled."

Car Computer

Ours was good, but Derek Knaggs really has built the real deal

Maker Profile

Derek Knaggs

Managing director

Derek Knaggs runs Flamelily IT, an IT supply and support company, and is studying Computing at the University of Worcester.

Find out more:
flamelily.co.uk

Take a look at the car computer that we make on page 106. Derek Knagg has already beaten us to it, and he's embedded the display in his dashboard and extended the setup to include screens for the rear passenger seats too.

"I removed the DVD player, which was a standard Ford head unit," explains Derek, "and then purchased a head unit from Xtrons. It's designed for the Ford Focus so it was a straight swap. The Xtrons radio has an S-Video input and that goes into the radio, so the Raspberry Pi displays as Auxiliary Input. There's two Auxiliary Outputs on the radio, so the Raspberry Pi sends a video to the main radio which then sends it back out to the screens in the passenger seats. What I've done is put in a device – like a VGA adaptor: it takes one input and puts seven out – that gives me the ability to have the Raspberry Pi running to the back screens on their own, so the radio can then control itself. I can have my kids watching movies at the back with the Raspberry Pi using an audio splitter (they've got headphones on), and we can be at the front using the normal radio controls, like the satnav for example. So that works well."

Above In this setup, the Pi is one of the inputs for the Xtrons head unit

RetroNES

Now you're playing with power. Raspberry Pi power to be exact, situated inside an old game console

Right It's difficult to see, but there are some difference to this NES compared to an original

Maker Profile

Chris Crowder

Programmer and database administrator

Working in the car industry, Chris develops manufacturing systems for production floor systems using .Net and SQL. In his down time, he likes to play videogames and tabletop games, but was previously limited to his PC for the former.

Find out more: **imgur.com/a/KPi2n?gallery**

> "It was a little intimidating at first as I wanted to make sure that this project looked and felt like an NES"

"It all started when my wife asked me what I wanted for Christmas", said Chris over email. "I had absolutely no idea but I had been wanting to mess around with a Raspberry Pi since it came out, so she got me a starter kit.

"While waiting for Christmas I started narrowing down ideas and found the RetroPi project. I thought that I would just install that, load some ROMs and call it good, then I remembered that I had some old NES and SNES controllers in storage. I went to get them and found my old childhood NES console along with the controllers. Once I got the NES back in the house and started looking at it, I found that almost all of the internals were damaged due to insects and moisture. All of the connectors were corroded and some of the boards had traces that were peeling. That is when I decided that I would use the Raspberry Pi to 'resurrect' the NES."

Above Everything is packed inside the original case, without needing to open it up to use it

Chris completely gutted the case and replaced the insides with a Raspberry Pi, hooking up I/O ports to the original connectors for the controllers and the AV cables and such. What's it like taking on a project with one of the most revered consoles in videogame history?

"It was a little intimidating at first as I wanted to make sure that this project looked and felt like an NES but with more flexibility. The biggest issue I ran into was that I wanted it to be able to work like an NES, meaning that if someone wants to play a game that they just turn it on, select a game and then they are playing. When they are finished all they have to do is press the power button to turn the console off. We can't do that easily with a Raspberry Pi since there is no ATX-style power switch. I was able to solve this issue with a Mausberry Circuit and a Python script. When the power button is pressed it communicates with the Pi via a GPIO connection and it runs the shutdown command. Once the Pi is shut down, the circuit cuts the power to the Pi."

The final product works great, with Chris reporting he can play on Atari, Sega and Nintendo games just fine. He's now looking to upgrade it with a Raspberry Pi 2 and increase the number of games he can play.

Refitting a NES

Do you fancy taking on the challenge of bringing your old console back to gloriously pixelated life? Thought so. In that case, you'll be needing this – here's a list of the equipment that Chris used to repair and revive his childhood NES console:

- **A broken NES console** – please don't do this to a working console
- Replacement **NES Door**
- **Canakit Starter B+** – soon to be replaced with a Raspberry Pi 2
- Panel Mount **Ethernet Cable**
- Panel Mount **HDMI Cable**
- Panel Mount **USB Cable**
- **USB A Male Connectors** 10pk
- **SNES USB Controllers**
- **Anker 13000 mAh 3 watt Battery** – this is for when there is no power outlet nearby or you want to be portable
- **Mausberry Circuit** – shutdown circuit that uses your own switch, USB
- **LEGO** – used to hold the Mausberry Circuit
- **Gorilla Glue**
- **Blue 3-volt LED** – the original NES LED was 12 volts, plus all of the other items in Chris' entertainment console are blue

Mission Control Desk

Astronaut training begins early in Jeff Highsmith's home, with his sons running launches from their home-made Mission Control

Above
The Mission Control desk groups the various functions into 'station' panels

Right Outside of playtime, this is just an ordinary homework desk. Almost

Maker Profile

Jeff Highsmith

Tinkerer extraordinaire

Jeff Highsmith loves to make new and novel things. The medium isn't important, and he enjoys scrounging for materials and making do with what's at hand.

Find out more: **jeffhighsmith.com**

Astrocarpentry

The panels are assembled from bare components. Jeff ordered the switches and then designed the panel layout and labels on his computer, printing them on inkjet transparency – clear acetate – and then gluing those onto some fibreboard that he had spray-painted a metallic grey. He estimates that the whole budget was $700.

Above These reflect the real stages of a NASA mission and play authentic recorded sounds

Jeff Highsmith is probably the best Dad in the world. Not content to just build his son a desk for his room, he modified it so that space adventures can start with the push of a button.

"My eldest son was starting kindergarten and he needed a desk to do his homework on," Jeff tells us, "and since I like to build stuff I thought I would make him a desk rather than buy one, and I was thinking, 'What would make a really awesome desk?' Well, having lots of buttons and switches like a mission control desk! Carpentry-wise it was pretty simple to build.

"The Raspberry Pi is up in the front-centre behind a piece of flexiglass, next to the Arduino that takes care of reading the inputs. The Pi handles all the sounds and the logic – the gameplay aspect. That was my first Python experience and it was pretty good.

"The desk has got several modular panels and each has a different function. So in the real mission control at NASA there's a desk that controls the retro stage, and for this desk I made a retro booster panel and put a bunch of rocket noises on it. There's a capcom (capsule communicator) panel, so you put on a little headset and you can talk to the astronaut that is in the spaceship in the other room.

"There are a couple of panels that have numerical displays: one reads out some attitude numbers, like x, y and z in space, and there's one that monitors the astronauts' vital signs (supposedly). There's one that does mechanical spaceship noises, like pumps, heating elements, buzzing noises, fans etc. I wanted it to be like you're turning things on and off, not just pushing a button that plays a sound – that's why I have the toggle switches as well as push buttons. There's a spot for the iPad in the middle too – you can watch videos of rocket launches.

"There are homages to actual NASA emergencies, like the stirring of the oxygen tanks that led to the Apollo 13 explosion. I have a switch that makes it sound like it's stirring the tanks, then it makes an explosion sound and plays the audio from the astronauts talking: 'Houston – we have a problem'. There's a sequence panel too that has the different mission stages on it and each of those plays a real NASA soundbite, all the way from the launch to the landing on the Moon to the splashdown."

Spaceship Bedroom

After successfully accomplishing his desk mission, Jeff Highsmith set his sights higher

Mission control was but one small step. Next was the mission itself, as Jeff explains: "So my boys would hit the buttons on the desk and go through all those mission stages and run around with their toy rockets, but having the actual spaceship, I thought, would be cool. The spaceship has some panels similar to the desk, but it also has a small screen in there that goes to a video camera in the cargo bay. There's a motorised hatch on the side that you can open up by flicking a switch, and then the camera shows you a cargo bay with a robot arm inside it. You can't see the cargo bay when you're inside because you're laying on your back, but looking at the screen you can see it and the controls are in front of you. It really feels quite fun – I've got a little toy Hubble space telescope in there and I hung a piece of fishing line from the ceiling with a little bit of metal on it, and then I've got a magnet in the space telescope. So you take the telescope from the cargo bay with the arm, move it over and snap it onto the string that hangs from the ceiling – we call that orbit. Once it's in orbit you pull the arm back in the cargo bay, close the hatch and your mission is complete, you return to Earth."

Jeff's going to upgrade this awesome setup further. "Eventually," he says, "I've got some ceiling satellites planned, so I'll have them orbiting a track in one of the bedrooms and the iPad can monitor the different sensors on the satellites. The track will be a thin metal rod under the ceiling in an ellipse, and then each satellite will have a tiny wheel extending from the top of it, which has a very small gear motor on it, so it'll hang from the track on that wheel. The idea is that the kids can build satellites out of Lego, put them in the cargo bay, then winch them up into orbit."

Above You can see the robot arm reaching out of the cargo bay door to the left

> "Having an actual spaceship, I thought, would be cool"

Automatic Roller

Nearly a Rube Goldberg machine, we hope it plays 'Powerhouse' when used

Maker Profile

Emil Jaworski

Maker

Finding himself in a rented apartment for a few months, Emil has decided to upgrade it himself.

Find out more:
imgur.com/a/OYdPo

There are many schools of thought regarding your sleeping environment to help aid better and more restful sleep. No electronics in the bedroom, try and relax before going to sleep, take a cool or hot shower depending on the time of the year. Some people require pitch darkness to get a good night's sleep though, while others like to wake up when the sun rises as part of a natural body reaction. Emil likes to do both of these things with his Raspberry Pi that automatically shuts and opens his blinds at specific times of day:

"7:30: Press up button, wait 10 seconds (I have smaller windows on the bottom and bigger above them, so after these 10 seconds, shutters are going to be open only on the bottom ones), press stop button. 8:00: Press up button. 22:00: Press down button."

Unfortunately, it means he has some electronics in his bedroom, but whatever works for him.

RasPi Terminal

We've had lots of Raspberry Pi over the years, but what about Apple Pi?

Maker Profile

Austen Barker

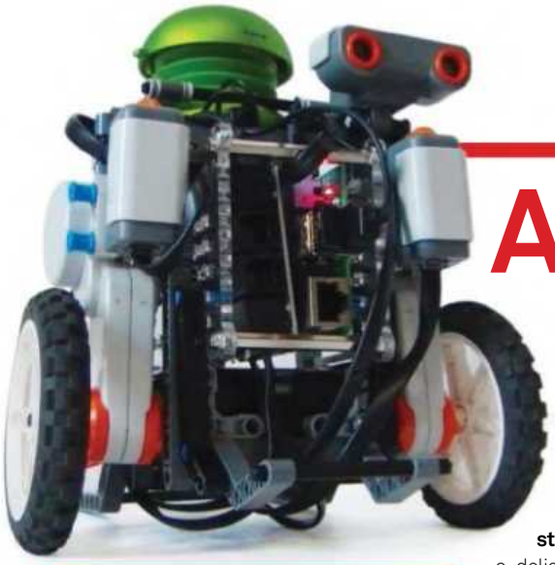
Engineering student

A Californian student that has a habit of messing around with any old computer hardware that he can get his hands on.

Find out more:
imgur.com/a/vOsML

"In my spare time I like messing around with older computer hardware that I come across. Originally I found the need for a terminal to connect over SSH to a server that I was using for an internship. It was more efficient than devoting a more powerful machine to it. I took a server to school with me to continue working for the same company as I studied.

I had a Raspberry Pi also sitting around and I started hooking it up to a monitor from an Apple IIc that I found at my university's electronics surplus. This became valuable when I started having to log in to a school server over SSH to compile assignments for my programming classes. Eventually, I found a keyboard from a Macintosh 512k and made it work over USB with a microcontroller and a custom wired key matrix, and I paired it with the monitor and Pi."



Alarm Clock Robot

Chasing your alarm clock may sound like a nightmare to some, but here it is in reality

When we talked about Rolly the alarm clock robot around the office, most people burst forth with a string of expletives not fit for print. It's a delightfully evil invention – an alarm clock you need to work for to turn off. It sounds like a great invention, getting people who have trouble waking up to actually get up out of bed and start the morning.

Like any other Dexter Industries robot, it runs on BrickPi, the LEGO Mindstorms adapter for the Raspberry Pi that enables it to interface with LEGO kits via programming.

"Today almost everyone uses their phone as an alarm clock, which has a range of benefits," the website explains. "Phones are easy to set, easy to adjust, play custom songs and can even sense when is the best time to wake you up. The problem is, unless your phone is across the room, we use our phones so much we can literally use them in our sleep. Why not build a robot that is able to do all these things, but won't stop till you get up and start moving! Our robot will be able to easily move randomly around the room over any surface, playing a custom alarm tone.

"In order to set the alarm, link the program on the BrickPi to your Google account and it will search events with the title 'wake1' and automatically start the alarm at the event's time. The alarm time can be adjusted using any device that can access your Google Calender."

Maker Profile

Taryn Sullivan

Advisor

Taryn is an international businesswoman. As well as flying between Shanghai and DC for her own engineering business, she now works with Dexter Industries to promote robotics education.

Find out more: **bit.ly/1BTYljv**

Robotics education

There are many ways to try and get kids excited with coding, like making games in Scratch, modifying *Minecraft* and teaching via robotics. The latter is a newer concept but still has the same merits – a physical creation that children are excited about then react to the programming they've done on it. Visible results and instant gratification is a great way to get imaginations fired up.

PiFM Radio

Sometimes the simplest hacks can open up whole worlds of possibility

Maker Profile

Oskar Weigl

Electronics engineer

Oskar is an electronics professional and hobbyist, as well as an avid forward and reverse engineer.

Find out more: **bit.ly/1om6BQE**

When studying at Imperial College London, Oskar Weigl and Oliver Mattos turned the Pi into an FM radio by connecting a wire (antenna) to GPIO 4 and using a custom Python module.

"There is a clock generation module in the hardware of the Raspberry Pi chip that lets you output a clock signal at a user-selected frequency," Oskar explains. "We used the DMA controller to send commands to the clock module to change the frequency and achieve frequency modulation. We had to overclock the clock generation module by a factor of 20 times. The sound is 14 bits per sample, enhanced to a higher number of bits using delta sigma modulation and the range is at least 50 metres."

Ras Pi Smart TV

Is your smart TV not smart enough? Open the case and put a Pi inside

Maker Profile

Tony Hoang

Graduate researcher

Tony Hoang is a PhD student studying computational biophysics and single molecule research at SUNY-Albany in Albany.

Find out more: **linkedin.com/in/tonyphoang**

"There's plenty of room for additional electronics inside the Hisense LED smart TV," begins Tony Hoang. "There's a large flat area for electronic parts in the centre of the TV where I placed my Raspberry Pi. The dual down-facing speakers were quite loud, so I removed one and replaced it with a USB hub. The back panel was mostly flat, so finding a spot for the LAN port and HDMI output wasn't too hard.

"The Raspberry Pi is powered by the logic board of the Hisense. There were the obvious 5v-500 mAh outputs from the 2x USB 2.0 ports, which I tried but I found out that the logic board shuts off the power to these ports when the TV turns off. To keep the Raspberry Pi turned on, I probed the logic board with a multimeter and found one from an unused GPIO."

Astrogun

An augmented reality light gun game

Walking around Maker Faire, you see some weird and wonderful things. At a Jerusalem Maker Faire you may have seen people wield a giant toy gun to shoot down virtual asteroids in Avishay's AR motion game Astrogun.

"In the Astrogun lies a Raspberry Pi computer," Avishay explains. "An IMU card connected to it (Sparkfun's MPU-9150 breakout board) gives it the ability to sense the unit's orientation. The Pi is then able to draw the elements seen from that angle. When the player moves, the graphics move, giving the 'object in the room' sense."

Why the Raspberry Pi? It was due to time, according to Avishay: "I had a short time to bring it to a working thing, so I had to pick a platform that was capable of the task and easy to use. The RPi fits that criteria. I used many software components designed for the RPi or tested on it – the Pi3D and RTIMULib. The combination of Pi as a hardware platform and Python as a programming language is the fastest way to materialise ideas."

Below Clever software gives you a window onto the Astrogun gameworld

Maker Profile

Avishay Orpaz

Electronics engineer

Find out more:
bit.ly/1AYPSqg

Pirate TV

Go all the way and totally rebuild Android TV!

Maker Profile

Donald Derek Haddad

Software engineer

Donald is an open source hacker.

Find out more:
donaldderek.com

The perfect companion to the Raspberry Pi Smart TV hack (opposite), Donald Derek Haddad's project is a custom TV interface that you can make yourself.

"Pirate TV is a smart television application that runs on the Raspberry Pi with the Raspbian OS," Donald tells us. "It's built with open source tools and shipped with a free remote controller, your mobile device. At its core lies a Node.js application that runs a web server with Express.js/Socket.io to handle users requests from the remote and trigger shell scripts. The TV user interface is rendered on a Chromium instance in kiosk mode. Videos are streamed from YouTube or other channels played on OMXPlayer, and cached including 1080P HD content. This project is a work in progress and it's not going to be able to tap into a lot of the content, which makes a Google (now Android) TV or other commercial platforms so valuable." Check out Derek's tutorial: **bit.ly/1l6YKpj**.

Pye Radio

Old meets new in this modified radio

Maker Profile

Tony Flynn

Senior embedded systems design engineer

Find out more:
bit.ly/19zrgPl

Upcycling is a great concept: recycling a product using new technology to make it relevant in the modern world. While standard analogue radio isn't dead yet, its nice to have options when listening to music. This is where Tony's idea came in:

"I'd been working on a Raspberry Pi to play music streams through my stereo. Once this was running I integrated the Raspberry Pi into an old radio named a Pye! With a background in woodwork and engineering this seemed like the perfect project."

However, this project wasn't a walk in the park for Tony: "The hardest part of the conversion is linking the tuner knob to the rotary encoder. For this radio I used the spring from an old biro as a drive train to link the tuner knob spindle to the rotary encoder through 90 degrees."

Did Tony have qualms about heavily modifying such a classic design? "None whatsoever! Some people don't like the modern style now stamped on this old radio, I think it's a new era, new look!"

Right The body has also been cleaned and repainted

Digital Camera Conversion

Classic aesthetics with modern convenience, this old-school camera has been upgraded with a Raspberry Pi

Maker Profile

Pete Taylor

Web manager

Pete works for a charity and has always tinkered with computers, ever since he got a hand-me-down BBC computer. He likes that the Raspberry Pi returns to a time when you could hack your own computer without making an expensive mistake.

Find out more: **bit.ly/1MKRASw**

> "The Raspberry Pi is a maker's dream – it's cheap and cheerful, and the community built around it is a brilliant resource"

Old cameras have a very specific design aesthetic that it seems has been lost to time, although nostalgia for them is still very strong in certain circles. Unfortunately, while nostalgia, desire and working cameras still exist in the 21st Century, usable film is quickly dwindling in supply. So if you like the aesthetic and aren't too bothered about using the old photo process, why not upgrade the insides with more modern technology?

"I wanted a suitable case for the Raspberry Pi camera board," Pete tells us, "and the Holga seemed a perfect fit. Most of the available cases are either a bit ugly or suited more towards stationary webcam-type applications."

Why the Raspberry Pi, though?

"The Raspberry Pi is a maker's dream – it's cheap and cheerful, and the community that's built up around the Pi makes a brilliant resource when you're stuck with a problem or want to find out more."

The entire build doesn't require a massive amount of components either. As well as the actual camera and Pi itself, Pete used a Raspberry Pi camera board to actually take photos, a Wi-Fi module for connecting to it remotely, a battery and a switch for it, and a few buttons and resistors to wire up the camera's control buttons to the Pi.

"It works better than I expected!" Pete said about the quality of the finished product. "Although it seems a bit daft to build a camera that's about as good as you'd get from a cheap camera phone, it's changed the way I take pictures. By removing the instant replay – most people seem to view the world through the displays on their phones – I can concentrate on taking the photo. Only seeing the pictures when I've taken the camera home and downloaded them to my PC adds a bit to the film nostalgia and I'm often surprised by the photos I've taken. Plus I've received some nice feedback about how the camera looks."

This is only the first iteration, and Pete has plans to make the next build easier and also use the original lens with the Pi camera.

Do it yourself

Version two of the project will result in a kit that people can use to convert their own cameras "that doesn't require you to take a Dremel to the insides of a Holga!". He's not decided yet on whether to make a kit that converts a Holga, or a kit that builds a Holga-esque case around the Raspberry Pi itself. Either way, the whole thing should also have a better photo-taking capability, which is the ultimate goal.

Top-left The camera of the past, updated for today

Left The Pi is the perfect size to fit inside the camera case

Maker Profile

Ian Cole

Maker and geek raiser

Ian Cole is a keen maker, hacker and inventor, and regularly blogs about his family projects with his two sons. Their Fireball project, for example, grew out of an innocent 'Can you make it playable?'

Find out more:
raisinggeeks.com

Fireball Pinball

Ian Cole and his sons enter the high score boards with Fireball HD

At raisinggeeks.com, Ian Cole and his two sons love a challenge. So the chance to repair a game table gave them the perfect Raspberry Pi primer.

"We've taken an existing pinball machine playfield," says Ian, "and built a new game from it. This required learning the underlying hardware first. Then we learned how to use the Raspberry Pi, pygame for sound and text graphics, and omxplayer for HD video, and we connected the software tools with the hardware of the pinball machine.

"We built a MOSFET circuit on a breadboard to test a single solenoid. When that worked, we duplicated it onto a hand-soldered protoboard and extended it to control the five solenoids. The Raspberry Pi handles graphics, audio, scoring rules, saving scores, etc. One Arduino drives the lamp and switch matrix, another drives the solenoids. The three are connected with an I2C bus."

Bitcoin Pool Table

Modern pool pros pay digitally by scanning a QR code to insert Bitcoins

Maker Profile

Stuart Kerr

Technical director

Stuart Kerr is the technical director of Liberty Games, which specialises in classic table and bar games. You might have heard of some of their hacks.

Find out more:
libertygames.co.uk

Liberty Games loaded a Raspberry Pi into the side of a pool table that enables people to make payments via a Bitcoin app on their phone to release the balls.

"We did this with the Raspberry Pi 1", Stuart tells us. "We tried to install the entire Bitcoin client on the Pi but it was struggling. Downloading the whole blockchain caused issues. It was going to be connected to the Internet, so we offloaded the heavier work to a server. The server receives the Bitcoin payment, communicates to the Pi securely that the payment has come in, and it syncs up the prices as well.

"We connected a PiFace via a breakout board that's monitoring on WebRCT for the go-ahead from the server, and once it gets that, it sends the physical voltage to the electrical ball release mechanism."

Above The price of a game is displayed, in Bitcoins, on the PiFace display module

LED Voice-Controlled Coffee Table

Light up your living room with a voice-controlled light show for a coffee table

Maker Profile

Mikel Duke

Software developer

Mikel loves programming, photography, biking and hiking, and sharing his projects.

Find out more:
bit.ly/1F0HDkW

Mikel found himself in a quandary: he didn't have a coffee table. However, instead of just heading to Ikea and picking up a LACK, he decided to make one himself. Not just any coffee table though, one that lit up using a Raspberry Pi to control the sequence.

"I used the Raspberry Pi because it was pretty easy to work with and has a great community. Whenever I run into an issue, there is always some documentation on how to fix it. I wanted to have network support and do more complex operations, like loading images, than I could do just using an Arduino with Ethernet. The combination of the two really made development faster. Now that Pi4J has added support for the Raspberry Pi's integrated SPI, I am working on controlling the LEDs directly, without the Arduino."

Recently, Mikel added voice control to it using Google Now. "After you say a command to Google Now, Autovoice, a plugin for the Android app Tasker, intercepts the response back from Google. If the response matches a Tasker task, the task will be launched. I have a few tasks set up to make web service calls, basically calling a URL with parameters. This URL connects to a Java based web service I created, running on my Raspberry Pi."

Apparently Mikel also encourages users not to use coasters, as drinks light up well when the table is switched on.

Project Jarvis

The 1950s future today with your voice-activated smart home automation system

Maker Profile

Mayur Singh

Computer systems engineering student

Located in South Africa, Mayur's expertise is in embedded systems and he is familiar with microcontrollers and systems that enable outside hardware interaction, such as the Raspberry Pi and the Beaglebone Black.

Find out more: **bit.ly/1Evlmci**

Home automation has been a thing for years now, with a fair few intrepid engineers and DIYers modifying their home. With wirelessly controlled lights, heating, garage doors and many other household items, it's amazing what some have achieved. Science fiction has always portrayed houses with advanced systems and Mayur has been busy creating such a system.

"The major inspiration came from watching the *Iron Man* movies, which is where the name Jarvis comes from. I decided to build a simple home automation system, however over the many years and many reinstalls of Windows, I lost my program. I built a new and improved home automation system, which features an AI assistant and more functionality, after I saw the third *Iron Man* film."

While home automation is one part of the project, convenience isn't the only factor.

"The main function of the system is to help save energy in homes," Mayur tells us. "Jarvis can read and monitor the electricity usage per light or appliance and this lets the AI perform certain tasks. These tasks are determined by outside factors, like whether or not a light should be switched on if there is adequate natural light in the room. This is a basic example but other factors influence the determination of the AI, which has control over the power to each light or appliance. The data is logged throughout the month and the system uses that information to achieve better results the following month. This is smart home automation."

The Jarvis AI is a major part of it, and you can talk to it. "He can control your home using voice interaction and make basic decisions about energy savings in a room. I have also recently built a wall-mounted tablet system that links to a local online grocery store. It uses voice control to identify what products you need and adds the items to a list until you specify the delivery."

With a year and a half of work on the project, everything Mayur has completed works reliably, but there are still more functions he wants to add.

Above The Jarvis interface is gorgeous, but you can control the system from anywhere using your smartphone or your voice

Above Alarm mode can only be disabled with the right finger or the master code

The voice

The Jarvis that this project is named after is well-known for its snarky remarks and tone of voice. Choosing an actual voice that works though is a hard enough task without going for a certain style, as Mayur explains:

"It's been difficult trying to find voices with a good API and price but I have settled for a free API which offers cloud conversion and just sends an MP3 sound file back. It takes longer but it ensures that the voice works on all OSs."

If you're interested in having a go at creating your own Jarvis-esque home automation system, check out the previous issue of **Linux User & Developer**. Joey Bernard, who writes the Python Column, began a three-part series (continued on page 72) that explains how to set up a digital assistant that uses Pyaudio and a few voice recognition modules to parse your spoken commands.

Popper buttons Beneath the fingers are small metal discs that can be activated just by a touch. Each button has been given a new function

Camera module This has been embedded into the fabric on the back of the glove, so photos can be taken by raising a hand and touching a button

Power pack As well as the Model A+, the USB charger used to power the Pi can be mounted on the glove by sliding it inside on the palm side

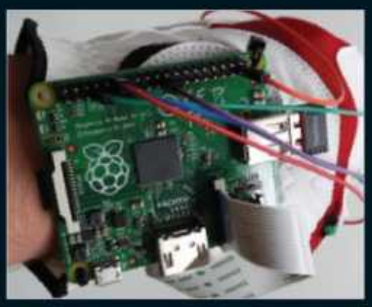
Raspberry Pi Using the Model A+, the Ras Pi could be mounted on the back of the glove. The old CAT5 cables were replaced with fewer wires

Components list

- Raspberry Pi Model A+
- Raspberry Pi camera module
- 5 Short cables
- 5 Popper buttons
- USB mobile phone battery
- USB Wi-Fi dongle
- Energenie add-on board
- Golf glove

Left For the next version, Dan may add a palm button for a hierarchical menu structure to navigate the functions on each finger-button

Below Now he's done the social media (in issue 148) and the home help modules, Dan will make a start on the fitness module



Pi Glove 2

Dan Aldred's new home help module controls lights, sends texts and can read out the text on photographs

What physical modifications have you made since we last spoke?

The glove is more portable – previously, the Raspberry Pi was in the wearer's pocket and you had the long CAT5 cables attached the glove. This has all been stripped back and several steps enabled this. Firstly, was the use of fabric clothes poppers to replace the tactile switches. These are metal and when one makes contact with a ground popper on the thumb, it creates the circuit. It has meant that the same functionality is achieved with five wires as opposed to the previous ten wires. Secondly, I have moved from a B+ model to the A+ model, which has meant that the Raspberry Pi is now small enough to be mounted on to the glove itself. Now the wires only need to run from the fingertip to the wrist. The camera module is also embedded within the glove. The lens is exposed through the glove but the rest of the camera is now housed within the fabric of the glove. You have to be a little bit more careful when you take the glove off, but the overall pay-off is that the glove is now lighter and more compact. The power comes from a small USB mobile phone charger which gives about six hours running time, depending on how much you use it for.

What new functions does the rebuilt glove have?

It was always the plan to develop the Pi Glove with 'modules' for various uses, starting with home assistance. Imagine if you did or maybe do struggle with a disability and you wake up in the night – the Pi Glove now enables you to check the time, which is read out, and a light can be turned on with a simple finger movement. If required, an emergency text can be sent to a carer, family member or the provider of other medical assistance. The fourth button enables the Pi camera, which takes a picture of a sign, for example. OCR is then used to recognise and store the text, which is then read back to you.

I decided to add the Pi camera around the back of the hand area – this linked in well, enabling a more mobile use of the camera; it can now be positioned in a direction that the user wants, is more accessible and you could even take a selfie! The main reason for this change was to enable 'on the fly' optical character recognition. I installed a Python OCR library and, combining this with the image taken from the Pi camera, the software can identify the text within the picture. This text is then fed back to the console and easy-Speak reads out the text. I tried various file formats – JPG seemed to work well. Also, changing the picture to black and white to pick up detail and differentiate between the text, had improved results. There were issues with it not identifying text in low light, and also if the text was the wrong way round or upside down. Finally, changing the saturation and increasing the sharpness produced usable results.

The emergency text on the second button makes use of Twilio, the web-based communication API, which enables the user to send a pre-written message requesting assistance. This could be adopted by others, such as the police or fire brigade, for use in dangerous situations. The current time is also added to the text.

To turn the lights on I used an add-on board by Energenie. Once installed you can use it to control up to four simple Energenie radio-controlled sockets independently, using a small program. The add-on board connects directly to the GPIO, which can be controlled as either input or output lines under your software control. A Python library is also available to program the sockets. I simply mapped the 'on' state to the click of the button and it turned the light on – from your fingertips!

Are you currently developing any new modules for the future?

The current module I am working on is fitness-related and will let the wearer take their heart rate, change their music and upload the details of a run to their social media site. This will be on the fly with no need to stop running or whatever sporting activity you are doing. Just touch the buttons and your workout music changes, or your heart rate is read and then converted to a string and read back to you through your headphones. I find that current apps on phones and watches disrupt your workout – I don't want to have to stop running to check my pulse or change the music.

I was looking at the idea of linking the glove functionally to a smartphone but this, I feel, would be moving away from the original aim, which was to remove the cumbersomeness of the phone – having to unlock it, load an app, wait for the camera to prepare itself. The glove enables you to click and go.

What would you like to do for the third iteration of Project New York?

I was introduced to the Micro Python Pyboard (**micropython.org/live**), which has a range of features built-in and is smaller, making it more suitable as a wearable component. The Micro Python board is a small electronic circuit board that runs Micro Python on the bare metal, and gives you a low-level Python operating system that can be used to control all kinds of different electronic projects.

The battery size is also an area that could be reduced – I am looking into this. The smaller these components are, the more natural the glove will feel.

> The Pi camera takes a picture of a sign. OCR is then used to recognise and store the text, which is then read back to you

Like it?

To learn more about the redesigned Raspberry Pi Glove and the current home help module, check out Dan's YouTube video (**bit.ly/1HVQTYA**) and the project write-up (**bit.ly/19xgQyC**).

Further reading

If you're interested in setting up optical character recognition for your own sign-reading Python projects, check out Dan's guide over at TeCoEd (Teaching Computing Education): **tecoed.co.uk/python-ocr.html.**

Build a Pi Glove – part 1

Create your own Wearable Tech glove that places control at your fingertips

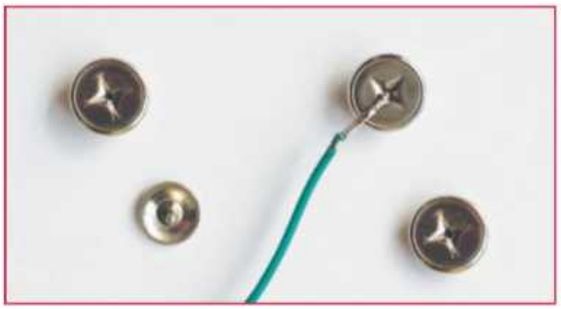
The Pi Glove is our project name for a wearable, social media-controlling glove – we're going to show you how to build it and how to program it. With the advent of Google Glass, Android and Apple smartwatches and various other disruptive technologies, the devices we're beginning to use today point towards a future where we may well one day wear all of the gadgets that enable us to communicate with each other (before they are simply embedded into our bodies...). The more natural or ergonomic this medium becomes, the more likely we are to use it, so we're going to use a Raspberry Pi to create a powerful yet comfortable glove. Part one of this two-part tutorial covers the hardware setup so that you can make your own version of the Pi Glove. Part two will cover the software to control music, a camera and social media using your glove's buttons.

What you'll need

- Raspberry Pi A+
- Clothes poppers
- USB Wi-Fi dongle
- Small cable ties
- Glove
- USB battery charger
- Pi camera module

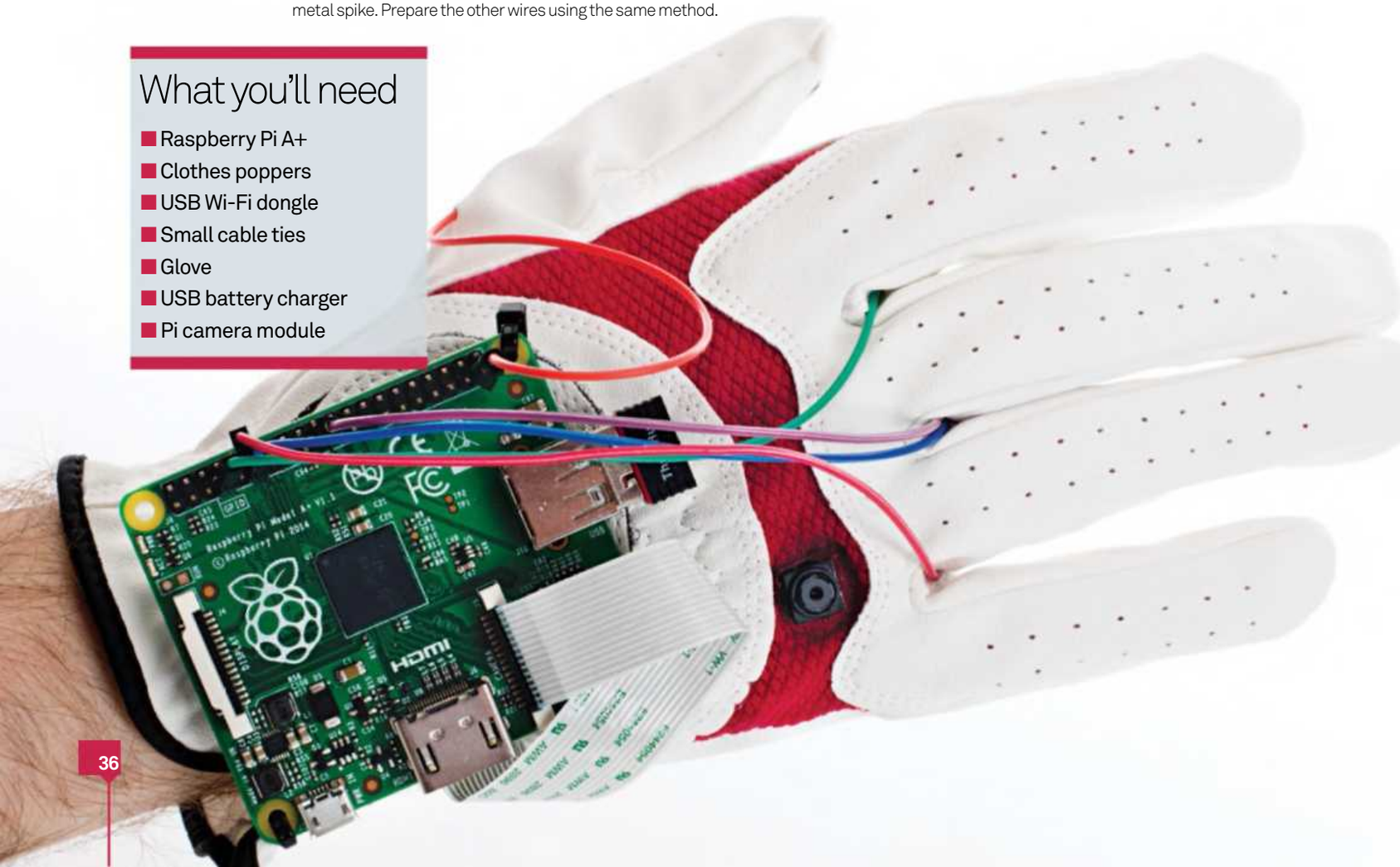
01 Strip and prepare the wires

Take the female-to-female jerky wires, select one end and remove the plastic coating; this can be done by applying a little pressure on the plastic cover – you can use your teeth but it's better to use a set of pliers! The end will now consist of a small metal spike. Prepare the other wires using the same method.



02 Attach the wires to the poppers

The next part is a little fiddly, so take your time as you work through each of the wires. First, you need to connect the wire to the popper whilst pinning the other end through the finger of the glove. Once through, close the two sides of the popper together, ensuring that the wire is secured in-between. This is easier if you turn each glove finger inside-out and then push the popper through. The key part of this step is to ensure that the wire stays connected.



Full code
FileSilo.co.uk

Left We placed our camera on the back of the hand, but you can put yours anywhere you like

03 Check the wires are in place

One wire/popper combo is attached to each of the four fingers and the thumb of the glove. The thumb forms the earth, or ground, of the connection when it comes into contact with one of the other metal finger poppers.

04 Add the Pi camera

The Pi camera is mounted into the glove, which enables you to angle your wrist and take a picture in the direction that the hand is facing. Make a small hole in the top of the glove, big enough for the camera lens to push through. On the Pi camera board you will find four small holes, so use these and the tiny cable ties to secure the camera onto the fabric of the glove.

05 Attach the Pi camera cable

Flip the glove inside-out so you can access the back of the camera. Take the camera connection wire, with the blue side facing away from the camera, and attach the wire to the back of the camera. On the Raspberry Pi, slide up the black camera holder piece and secure the wire into place. You may wish to thread the wire through the side of the glove to conceal it.

> The Pi camera is mounted into the glove, which enables you to angle your wrist and take a picture

06 Checkpoint

At this stage you have the five poppers attached individually to a wire and through each of the individual fingers, and also attached to the thumb. Each of these wires runs through the inside or outside of the glove to the Raspberry Pi. The Pi camera is also attached to the glove, with the lens exposed to enable pictures to be taken.

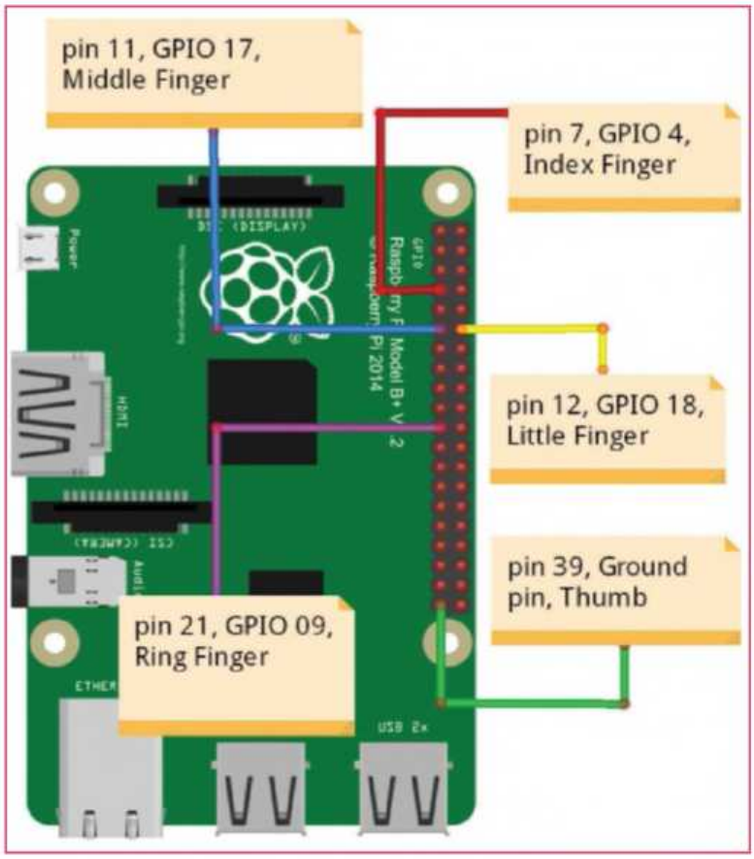
07 Mount the Raspberry Pi

You can mount the Raspberry Pi onto the top of the glove using the small cable ties and the four holes located on each corner of the Pi – this makes the overall glove more portable. Take your Pi and orientate it so that the SD card slot is facing away from your wrist. Thread the cable ties through the holes and through the glove, securing the Pi in place. Be careful not to over-tighten the cable ties.

Wpa_suppliant

Wpa_supplicant is a background program that runs and acts as the component to control wireless connection. It supports both text and GUI interfaces.

On saving the file, wpa-supplicant will normally notice that a change has occurred and try to make a connection to the network



Above It won't fit onto your glove as neatly, but you can use the Model B+ or 2B as well as the A+

08 Attach the wires to the GPIO pins

Now that all the hardware is attached to the glove, the next step is to connect the finger and thumb wires to the GPIO pins on the Raspberry Pi. The pin numbers we use here identify the physical pin number on the board. Take the thumb wire and connect it to physical pin number 39 – this is the ground contact for the other buttons. Now take the index finger wire and connect to physical pin number 7.

09 Connect more wires to the GPIO pins

Using the same method as described in Step 8, take the middle finger wire and connect it to pin number 11, the ring finger to pin 21 and finally the little finger wire to pin 12 – again, these numbers are the physical pin numbers on the board.

10 Boot up the Pi

The first time you boot up your Raspberry Pi you will need to set up your Wi-Fi connection. Connect the Pi to your HDMI monitor and insert the Wi-Fi dongle. If you are using the newer A+ model, which only has a single USB, you may require a USB hub to enable you to add a keyboard and mouse. You could always set up the Wi-Fi on a B+ model, which offers more ports, then transfer the card to the A+. Add the USB mouse, keyboard and power supply. Boot the Pi up.

SSID

SSID is a case-sensitive, alphanumeric, 32-character unique identifier attached to the header of packets sent over a wireless local area network (WLAN) that acts as a password when a mobile device tries to connect to it.

11 Add Wi-Fi app

Open up the Wi-Fi Configuration tool from Menu> Preferences. When open, select Manage Networks and then Scan – this will find and list all of the available Wi-Fi networks in your area. Select your network from the list and double-click. You will then be prompted to enter your WEP code, so enter this in the PSK text box and press Save. Return to the Current Status page and click Connect. This will connect your Pi Glove to the Wi-Fi.

12 Set up Wi-Fi in the command line, part one

If you are using SSH to access your Pi, you can set up your Wi-Fi dongle from the command line. In the terminal window, type: **sudo iwlist wlan0 scan**. This will scan for available Wi-Fi networks. Look out for: ESSID:"test-network" – this is the name of the Wi-Fi network.

'IE: IEEE 802.11i/WPA2 Version 1' is the authentication that is used; in this case it is WPA2, the newer and more secure wireless standard which replaces WPA1.

You will also need your password for your Wi-Fi network – for most home routers, this is located on a sticker on the back of the router. The ESSID (ssid) for the network in this case is 'test-network' and the password (psk) is 'testingPassword'.

13 Set up Wi-Fi in the command line, part two

Now add your Wi-Fi settings to the wpa-supplicant.conf configuration file. In the terminal window, type:

```
sudo nano etc/wpa_supplicant/wpa_supplicant.conf
```

Scroll to the bottom of the file and add the following lines:

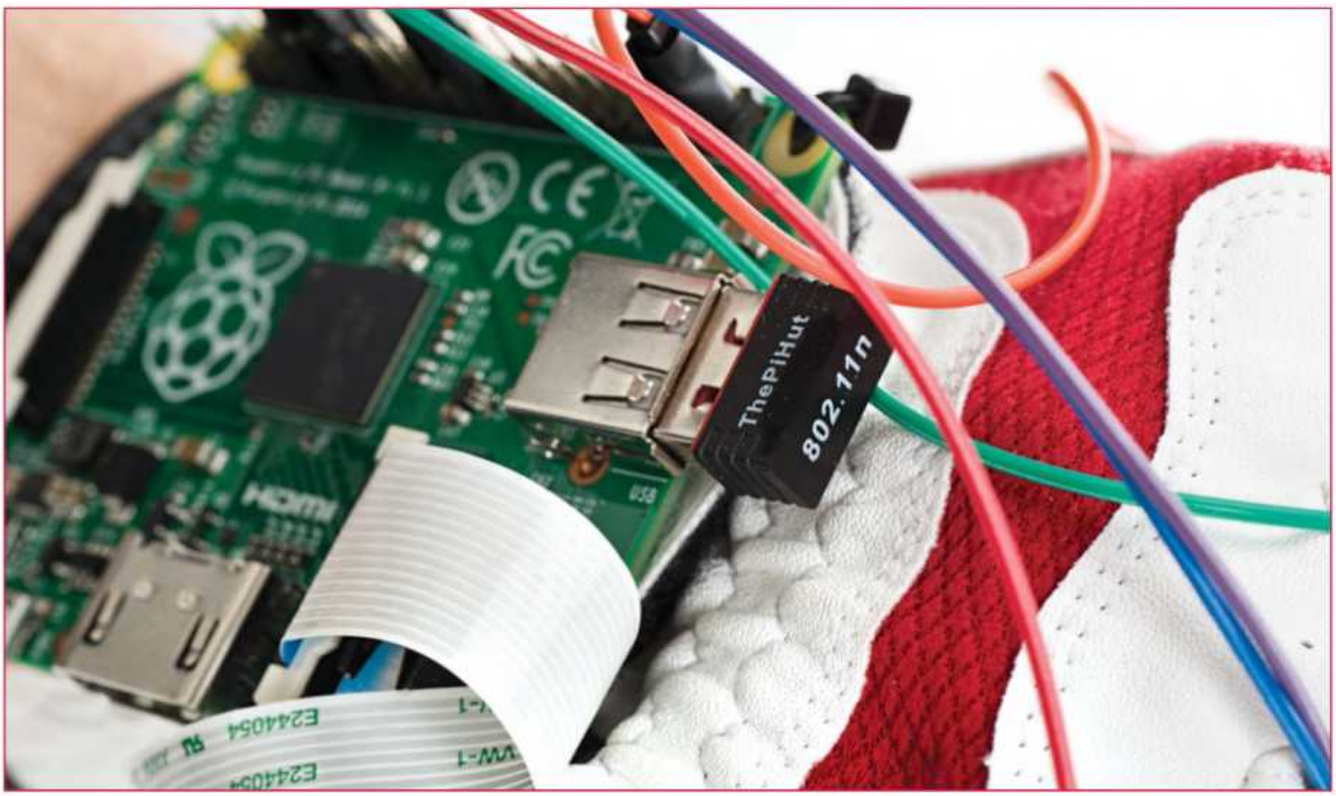
```
network={
    ssid="The_ESSID_from_earlier"
    psk="Your_wifi_password"
}
```

Using the example network found in Step 12, you type **ssid="test-network" and psk="testingPassword"**. Now save the file by pressing Ctrl+X then Y, then press Enter.

14 Set up Wi-Fi in the command line, part three

On saving the file, wpa-supplicant will normally notice that a change has occurred and so it will try to make a connection to the network. If it doesn't do this, you can either manually restart the interface – just run **sudo ifdown wlan0** followed by **sudo ifup wlan0** – or instead reboot your Raspberry Pi with **sudo reboot**.

To test that the Pi is successfully connected to your Wi-Fi, type **ifconfig wlan0**. If the 'inet addr' field has an address beside it, the Pi has connected to the network. If not, check that your password and ESSID are correct.



Above We're using the brilliant Wi-Fi dongle from The Pi Hut, which you can get for just £6: **bit.ly/1LfkCgZ**

Full code listing

```
###TeCoEd Test Version###
###Glove Button Test ###
###Project New York###

import time
import random
import os
import sys
import subprocess
import RPi.GPIO as GPIO
from sys import exit

######Set up the GPIO Pins ######
GPIO.setmode(GPIO.BCM)

###sets the pin to high ###
GPIO.cleanup()
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)  ##11 on the BOARD
GPIO.setup(18, GPIO.IN, GPIO.PUD_UP)  ##12 on the BOARD
GPIO.setup(9, GPIO.IN, GPIO.PUD_UP) ##21 on the BOARD
GPIO.setup(4, GPIO.IN, GPIO.PUD_UP) ##7 on the
GPIO.setwarnings(False)##switch off other ports

 while True:
    if GPIO.input(4) == 0:
        print "you pressed button one"

    if GPIO.input(17) == 0:
        print "You pressed button two"

    if GPIO.input(9) == 0:
        print "You pressed button three"

    if GPIO.input(18) == 0:
        print "you pressed button four"
```

15 Disable the Wi-Fi power management

If left idle, the Wi-Fi power management system on the Raspberry Pi may drop the Wi-Fi connection – this may, for example, occur if the glove is out of range of the router. To disable the power management, load the terminal window and type:

```
sudo /etc/network/interfaces
```

At the end of the block of code, add the following line:

```
wireless-power off
```

This will ensure that the Wi-Fi stays connected whilst in range.

16 A simple test program

Now you have completed the hardware section of the Pi Glove, you can use a simple program to test the connections and make sure that all of the poppers are working correctly and responding to the thumb and finger contacts. Download the test program from **FileSilo.co.uk**. Run **sudo idle** in a terminal to open the Python editor, then start a new file and insert the code. With your Pi Glove attached, save and run the program.

17 Run the code

The test program will respond to each connection of the fingers and the thumb, and display a message stating that the respective button has been pressed – like so: 'button one has been pressed', 'button two has been pressed', etc. If this fails, check for the following errors: 1) incorrect wiring on the GPIO pins, 2) loose wires not in contact with the poppers, and 3) thumb and finger not in contact. Part two covers how to develop a program that brings control to your fingertips.

Build a Pi Glove – part 2, creating software

Develop a program code to add functionality and features to your Pi Glove, bringing interactivity to your fingertips

In part one, we covered the creation and the hardware setup of a wearable tech glove formally known as Project New York. This tutorial shows you how to create a program and write the software to add interaction to the glove's buttons. Once you have created the basic code structure, you can develop new interactions by coding and adding your own functions. The program is coded in Python 2.7 to ensure all the libraries are compatible with the Raspberry Pi hardware and the Raspbian operating system. On completion, the glove will give you spoken instructions, tell you the current time, take a picture with the Pi camera module and play a random selection of music, all from your fingertips.

01 A quick test and recap

Ensure that your glove hardware consists of at least five wires connected to a Pi which is mounted to the glove. A Pi camera is also embedded or attached to the glove. Boot up your Raspberry Pi; this could be a separate Pi. Initially, it is worth running the test program below, to ensure that all the hardware and wires are connected correctly and working properly.

```
import RPi.GPIO as GPIO
######Set up the GPIO Pins ######
GPIO.setmode(GPIO.BCM)
###sets the pin to high ###
GPIO.cleanup()
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)
##11 on the BOARD
GPIO.setup(18, GPIO.IN, GPIO.PUD_UP)
##12 on the BOARD
GPIO.setup(9, GPIO.IN, GPIO.PUD_UP)
##21 on the BOARD
GPIO.setup(4, GPIO.IN, GPIO.PUD_UP)
##7 on the BOARD
GPIO.setwarnings(False)  ##switch off other ports

while True:
    if GPIO.input(4) == 0:
        print "You pressed button one"

    if GPIO.input(17) == 0:
        print "You pressed button two"

    if GPIO.input(9) == 0:
        print "You pressed button three"

    if GPIO.input(18) == 0:
        print "You pressed button four"
```

02 Install the Python libraries

Assuming all went well with the test, you are set up and ready to build the new program. The good news is that most of the modules that you will use in the program are already pre-installed on the Raspbian operating system. To add the 'spoken instructions' feature you will install a module called eSpeak. In the LX Terminal, type:

```
sudo apt-get install espeak python-espeak
```

To play the MP3s, syou will use a program called mpg321. To install this, type:

```
sudo apt-get install mpg321
```

Once installed, restart the Pi.

03 Test eSpeak

eSpeak is a simple, compact, open source software speech synthesiser that uses English and other languages. It works by taking a text string and then converting it into audio. But that's not all – you can also edit the voice, pitch, volume and other parameters of the speech. Test that it is working by creating a new Python file and using the code **espeak.synth ("This is a Test")**. Now, when you run the program it will read out the phrase "This is a test".

```
from espeak import espeak

espeak.synth ("This is a test")
```

04 Import modules

The Glove program uses a number of modules for its various functions. These are Python files that contain a number of definitions, variables, functions and classes. Import the modules below into your Python program – these will give your program access to the MP3 player, the Pi camera module, the GPIO pins and eSpeak.

```
import time
import random
import os
import sys
import subprocess
import picamera
import RPi.GPIO as GPIO
from sys import exit
from espeak import espeak
```

Twitter

The original Project New York Glove featured the ability to tweet the picture that was taken with the Pi camera. If this is a feature you are interested in, you can read more here about how to set up a Twitter API (**tecoed.co.uk/twitter-feed.html**). Button 4 also retrieved the train times between two stations then stored and read these out through the headphones; further details can be found over here: **tecoed.co.uk/scraping-trains.html**.

What you'll need

- Pi Glove or a similar hardware setup

05 GPIO pull-ups

To register that you have triggered the finger button, we make use of GPIO pull-ups to detect that the two contacts have touched together. The pull-up resistor sets the current to 0 volts. When the two wires connect, the voltage changes and this change in state is recognised, triggering the function which you will assign to each of the buttons. If you have no pull-up or pull-down then the GPIO pin can change state, for instance if there is external interference, and this means that it can trigger your button even if you did not touch it. To set these up, add the following code to your program:

```
GPIO.setmode(GPIO.BCM)

###sets the pin to high ###
GPIO.cleanup()
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)
##11 on the BOARD SPARE
GPIO.setup(18, GPIO.IN, GPIO.PUD_UP)
##12 on the BOARD MUSIC PLAYER
GPIO.setup(9, GPIO.IN, GPIO.PUD_UP)
##21 on the BOARD TAKE A PICTURE
GPIO.setup(4, GPIO.IN, GPIO.PUD_UP)
##7 on the BOARD TIME
GPIO.setwarnings(False)##switch off other ports
```

06 Add the spoken instructions

Since there is no visual display, you will not know that the program is running or that it is ready. Therefore, at the start of the program it reads out the button number and the function of each. This uses the same code from Step 3, calling eSpeak to convert the text to an audio wave and play it back through a speaker or a pair of headphones. You can customise the introduction and what instructions are given. Use **time.sleep(2)** to add a slight break between the sentences and make the speech more natural.

```
espeak.synth ("Welcome to the PI GLOVE")
time.sleep(2)
espeak.synth ("Please make a selection")
time.sleep(2)
espeak.synth ("Button 1 - tell you the time")
time.sleep(2)
espeak.synth ("Button 2 - take a picture")
time.sleep(2)
espeak.synth ("Button 3 - play some tunes")
time.sleep(3)
espeak.synth ("Please select your button")
```

Full code listing

```
import time
import random
import os
import sys
import subprocess
import picamera
import RPi.GPIO as GPIO
from sys import exit
from espeak import espeak

global File_Number ###number if photo
global file_name ###name of photo
File_Number = 1

######Set up the GPIO Pins ######
GPIO.setmode(GPIO.BCM)

###sets the pin to high ###
GPIO.cleanup()
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)
##11 on the BOARD SPARE
GPIO.setup(18, GPIO.IN, GPIO.PUD_UP)
##12 on the BOARD MUSIC PLAYER
GPIO.setup(9, GPIO.IN, GPIO.PUD_UP)
##21 on the BOARD TAKE A PICTURE
GPIO.setup(4, GPIO.IN, GPIO.PUD_UP)
##7 on the BOARD TIME
GPIO.setwarnings(False)  ##switch off other ports

###Introduction###
###welcome messages###

espeak.synth ("Welcome to the PI GLOVE")
time.sleep(2)
espeak.synth ("Please make a selection")
time.sleep(2)
espeak.synth ("Button 1 - tell you the time")
time.sleep(2)
espeak.synth ("Button 2 - take a picture")
time.sleep(2)
espeak.synth ("Button 3 - play some tunes")
```

OS in Python

The Python OS module enables you to interface with an operating system, which provides a way to use Python to interact with a Linux, Windows or Mac computer. Python code can then be used to control OS system commands such as changing file names, creating folders and files, as well as changing file paths. You can also find out information about your location or about the process.

07 Set up the time

At this point you are now ready to set up the function for the first button, which will tell you the time in a similar fashion to the old 'speaking clock'. This feature means you don't have to take out and unlock your phone – simply press the button and the current time is read back to you. Line 2 of the code creates and stores the current time as a variable

```
current_time = (time.strftime("%H:%M:%S"))
```

A second variable, line 3, is used to store the 'time message' which is then used by eSpeak to read out the time to you, line 4. Add the code to your program:

```
def what_is_the_time():
    #global time
    current_time = (time.strftime("%H:%M:%S"))
    the_time = "The current time is %s" % current_time
    espeak.synth(the_time)
    time.sleep(2)
```

08 Set up the camera

The picamera module is pre-installed on the Raspberry Pi, so you are ready to create a function which will trigger the camera and save the picture as a new file called newpic.jpg (line 5). The third line is useful to test that the camera is taking a picture and also to familiarise yourself with where you are pointing the camera. When triggered it will display a preview of what the camera sees on a connected monitor or television.

```
def take_a_pic():
    with picamera.PiCamera() as camera:
        camera.start_preview()
        time.sleep(2)
        camera.capture("newpic.jpg")
```

09 Take a selfie

It is possible to perform a test in order to ensure that the camera is working by calling upon the **take_a_pic()** function. Do this by opening up a new Python window and then add the previous code from Step 8. Save and run the code, and you should see a two-second preview of what the camera sees and then the camera will capture this image, which is then stored in the Pi/Home folder.

10 Save as a new file name

Currently, each time a new picture is taken, it overwrites the previous file. Annoyingly, this means that you will lose the last picture you took. To stop this, create a global variable called **File_Number**, line 1. This variable is incremented each time a new picture is taken. Create a second variable, called **file_name** (line 2) – this variable is combined with **File_Number** to create a new unique file name each time the picture is saved (line 4), preserving your previous pictures. Line 5 ensures that the **File_Number** value is incremented by one each time a photo is saved.

```
global File_Number ###number if photo
global file_name ###name of photo
File_Number = 1
```

```
file_name = "Picture" + str(File_Number) + ".jpg"
            File_Number = File_Number + 1
```

11 Final camera code

The complete camera code uses a function that combines the features from Steps 8 and 10 to trigger the camera and save the image as a new file with a unique file name each time the two 'poppers' connect. Add the code below to a new line underneath your **time** function.

```
def take_a_pice(): ###Takes a picture ###
    global File_Number
    global file_name
    with picamera.PiCamera() as camera:
            time.sleep(0.5)
file_name = "Picture" + str(File_Number) + ".jpg"
             File_Number = File_Number + 1
```

12 Save the music

There are two small steps to take before you can enable the music player. First, download a number of MP3s and save the file names as numbers – for example, 1.mp3, 2.mp3, 3.mp3 and so on. For the second step, create a variable at the beginning of your program to store the file names, such as:

```
songs_list = ["1", "2", "3", "4", "5"]
```

This variable is used to select the song.

```
###Code for MP3 random play list###
songs_list = ["1", "2", "3", "4", "5"]
```

13 The music player

Instead of creating another function for the music playback, the MP3 player is called directly from the GPIO pin 17 pull-up. It makes use of the variable **song_list**, which holds the file names stored as a list of numbers: 0,1,2,3,4,5. In the Home folder, you'll have your six music tracks named 0.mp3, 1.mp3, 2.mp3, etc. In order to make this a shuffle-based player, we can use the following line of code:

```
os.system('mpg321 '+ (random.choice(songs_list)) + '.mp3 &')
##change
```

... which calls the operating system to load the mpg321 software and select a random number from the play list, and it then loads and plays the corresponding mp3 file.

14 Stop the music

The code in Step 13 will keep the music playing continuously. To stop the music, use the code:

```
os.system('sudo killall mpg321')
```

Map this code to the button on the glove and, by holding down the button for a little longer, you can cycle through a variable called **song_play** (line 3), which changes from 'yes' to 'no'. When the variable is 'no' (line 9), a conditional is used on line 4 to check the state and then use the following code:

```
os.system('sudo killall mpg321')
```

... in order to stop the music playing (line 8). Listen to the spoken instructions, and you can then time it right to properly end the music. Now we've explained what's going on, add the following code into your program:

```
os.system('sudo killall mpg321')
        espeak.synth ("Music Player ")
        song_play = "yes"
        if song_play == "yes":
            os.system('mpg321 '+ (random.
                choice(songs_list)) + '.mp3 &')
            if GPIO.input(17) == 0:
             #turns off song longer hold
                os.system('sudo killall mpg321')
                song_play = "no"
                espeak.synth ("MP3 player stopped")
```

15 Create the button structure

Now you have created three features for your glove, you can start assigning them to the finger buttons, which will trigger each function when they are connected with the thumb button. This is achieved with a simple set of conditionals, like: **if GPIO.input(17) == 0:**. This checks if a GPIO pull-up is present, and then if so, it runs the assigned function. Add the four conditionals below into your program. Remember to move the music player code from Step 14 so it's beneath the GPIO pin 17 code.

```
while True:
    if GPIO.input(4) == 0:
    if GPIO.input(9) == 0:
    if GPIO.input(17) == 0:
    if GPIO.input(18) == 0:
```

16 Call the functions

Once you have created your conditionals to test for the button pull-up, the final part of your program is to add the function for each GPIO. For example, add the time function to GPIO pin 4 with:

```
if GPIO.input(4) == 0:
        what_is_the_time()
```

This will run the time function you created in Step 7 each time the pin 4 button is connected to the thumb button. You will also want to add some instructions under each button to inform the user what is happening. For example, when triggering the camera it is useful to know when the picture is going to be taken – have a look at the code example below.

```
time.sleep(1)
espeak.synth("Preparing the camera")
time.sleep(2)
espeak.synth("Camera ready, smile")
time.sleep(1)
take_a_pic()
###enables the camera def and takes a picture
espeak.synth("Picture taken and saved")
time.sleep(3)
espeak.synth("Press button two to tweet your
picture")
```

17 Other functionality

Save and run the program as the root user, then debug the code and test the contacts. Common errors may be incorrect wiring on the GPIO pins, loose wires not in contact with the metal poppers, or the thumb and finger not in contact. Once working, you can now create your own interactions for your glove – for example, turn lights on and off, send an SMS to a mobile phone, control your TV or read text in a text file. ■

> You can now create your own interactions for your glove – for example, turn lights on and off or send an SMS to a mobile phone

Full code listing (cont.)

```
time.sleep(3)
espeak.synth ("Please select your button")

###Code for MP3 random play list###
songs_list = ["1", "2", "3", "4", "5"]

def what_is_the_time():
    #global time
    current_time = (time.strftime("%H:%M:%S"))
    the_time = "The current time is %s" % current_time
    espeak.synth(the_time)
    time.sleep(2)

###Code for the Camera to take a picture###
def take_a_pice(): ###Takes a picture ###
    global File_Number
    global file_name
    with picamera.PiCamera() as camera:
        time.sleep(0.5)
        file_name = "Picture" + str(File_Number) + ".jpg"
        File_Number = File_Number + 1

while True:
    if GPIO.input(4) == 0:
        what_is_the_time()

    if GPIO.input(9) == 0:
        os.system('sudo killall mpg321')
        time.sleep(1)
        espeak.synth("Preparing the camera")
        time.sleep(2)
        espeak.synth("Camera ready, smile")
        time.sleep(1)
        take_a_pic() ###enables the camera def and takes a picture
        espeak.synth("Picture taken and saved")
        time.sleep(3)
        espeak.synth("Press button two to tweet your picture")

    if GPIO.input(17) == 0:
        os.system('sudo killall mpg321')
        espeak.synth ("Music Player ")
        print""
        song_play = "yes"
        if song_play == "yes":
            os.system('mpg321 '+ (random.choice(songs_list)) +
                                '.mp3 &') ##change the song!
            if GPIO.input(17) == 0: #turns off song longer hold
                os.system('sudo killall mpg321')
                song_play = "no"
                espeak.synth ("MP3 player stopped")

    if GPIO.input(18) == 0:
        print "Add your own button"
```

Clip-on The attachment clips on to your regular prescription glasses or sunglasses, so you don't have to worry about sacrificing your ability to see

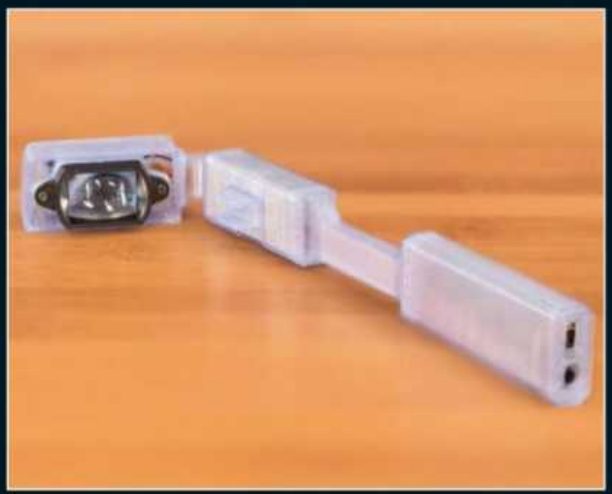
Video display The tiny screen on top of your glasses provides a 320 by 240 display and feels like you're looking at a 52-inch screen a few feet away

Casing All the components taken from the original video glasses have been rewired and then sealed in this eight-piece, 3D-printed enclosure

Power If you want to use Pi Glass on the go then plug it into a portable USB battery pack. The Ruiz brothers mounted it onto a belt: **bit.ly/1HDj1y8**

Components list

- Raspberry Pi Model B
- NTSC/PAL video glasses
- Miniature wireless USB keyboard with touchpad
- USB battery pack
- 3D printer
- CAD files (.stl)
- Composite video cable
- 30AWG wire wrap
- Heat shrink pack



Left The Pi Glass attachment is affixed to your regular glasses using the small clip on the inside of the central block

Below These video glasses cost about $110/£70 and, while not exactly suited for long-term use, they're perfect for repurposing into other optical projects like the Pi Glass

Pi Glass

Adafruit creatives Noe and Pedro Ruiz hack video goggles to make a 3D-printed Google Glass-like attachment

How did you get started with Adafruit?

Noe It was about a year ago, we came on their show-and-tell and we wanted to show people what projects we were working on. At the time it was a simple wearable project – a 3D-printed belt buckle with Adafruit LEDs and their GEMMA microcontroller, so the thing there is mixing self-expression and design with the electronics and making it flashy and cool. Adafruit really liked that, and what they liked even better is that we happen to make videos as well, so Phil [Torrone], a cofounder of Adafruit, wrote to us asking if we'd like to be an author on their Adafruit Learning System. We said sure, we can write documentation, but we can also do video too. And that sort of led to starting another project and it gained momentum from there, and every week we've been coming out with a new 3D-printed project since. Recently it's been getting so much bigger, and it's always a challenge because every week we're upping our skills – it's like, can we design that, will that print? So far it's been more successes than failures. We do a good job learning from the bad stuff and capturing the really good stuff and telling people in our guides how to keep moving on. It's very hard stuff but we try to make it look like it's not so hard so that people try it out and learn from it.

Pedro We try to make the guides as repeatable as we can – step-by-step guides that are easy to use.

Noe It's so open sourcey! We had no idea of the open source hardware movement and it's completely empowering to give away our design files. When we started out as Pixil 3D we really didn't give away our designs – we'd hold on to them because it was our stuff. But now it makes so much sense to give away our designs since you have that incentive to. It's like hey, here's a cool project idea – just buy some parts and then follow along with our circuit diagrams and tutorials.

Pedro It really speaks to what 3D printing is becoming; it's the shell that holds all the components that bring it to life inside.

So what exactly does your DIY Glass do?

Noe The idea was sort of inspired by Limor [Fried] herself – she has these hundred-dollar video glasses on the shop and she said: "You know what? Let's take it apart. Let's take the guts out, the actual circuitry, and make a new format for it. Instead of being two glasses let's make it clip on to your existing glasses kinda like Google Glass, but let's make it for the Raspberry Pi." So that was the original idea and it was rather simple because there really wasn't much programming or software involved – it was just repurposing this component inside this hundred-dollar pair of glasses and making it more Google Glassy and more DIY. From a design standpoint it was really challenging because the tolerances and things for that was kinda hard, especially for different machines – you're always looking at different tolerances; even when you're slicing it, things will come out a little bit differently. So that's why it's so important to give away the files and to tell people that you can modify it and you can make it work for you. And quite a few people have made their own and printed it for their application.

Pedro It's a good foundation for anybody who can build on top of it. So we've seen different Pi UIs that mimic the Google Glass UI – this is something that somebody could take and sort of adapt.

Noe It's a great example – so someone who's not super ace at designing or printing but maybe has the software chops can take this project and make it even better. We'd really like to see that.

How did you make the attachment after you split up the original video glasses?

Noe So I guess for starters we bust out the calipers and we start measuring like crazy. From there we designed the components, we remake them in CAD – our favourite CAD right now is 123D Design, which is from Autodesk. We use it literally on a daily basis. But you start off by making the components and modelling them out, and creating the enclosure on top of that and just chiselling away and creating features, figuring out how to split it up into pieces so that it can print without any support material. We really strive to make our designs with no supports – that way you can get a really clean design that looks beautiful and doesn't require that extra bit of waste. And it is 3D printing – it's rapid, right? So we prototype the piss out of our projects! We're so fortunate that we have the time to do it. It's hard to keep it under two weeks, but it really feels like it's a rush and we do step back and take the time to make sure it's right.

Pedro We always have a buffer of at least a month with projects already in the works and we make sure we keep the pipeline full – sort of like a TV schedule. And sometimes if we can't finish quite in time, we let people know and say hey, you can finish this for us by all means – go ahead and pick up where we left off.

How is the Pi talking to the attachment?

Noe It's just plugged in through HDMI, really – it's just an add-on to the Pi to make it mobile. You just plug into a battery bank, so it's really simple in that way.

Is there an easy way to control the output?

Noe We have a small wireless keyboard that we sell in the shop, so we thought we'd keep it as simple as possible and use that to control things on the Pi.

Do you think this is a project you'll revisit?

Pedro Well, with the release of the A+ we might revisit it in a future episode.

Noe Maybe something more enclosed and more specific to the Pi.

Pedro Yeah, so build more libraries for talking to sensors and things like that. We might try and incorporate some eye-tracking, things like that.

Noe We have a really cool remote team that does different projects as well, so we're just now starting to collaborate with them because they're more skilled and disciplined in software engineering, so it's really cool to bring those two minds together – the design and videography and then the software engineering.

Like it?

If you want to make your own Pi Glass then check out Noe and Pedro's tutorial on the Adafruit Learning System: **bit.ly/1fbHhfw**

Further reading

To see some of the crazy electronic costumes and 3D-printed gadgets that the Ruiz brothers make, check out their main page over at the Adafruit site: **bit.ly/1yBSECn**

VISUALISE MUSIC WITH LEDS

Get the party started with a five-metre LED strip that reacts to music

What you'll need

- LED strip
- USB sound card (we used a Behringer UCA202)
- Breadboard
- Female to male jumper cables
- 3 x TIP120 NPN transistors
- 3 x 220Ω resistors
- DC jack

In this feature, we will write a Python script that takes audio as the input, either as a line-in stream from an audio card or as a WAVE file to play. This audio stream will then be filtered into three bands – bass, middle and treble – using the signal-processing library that comes with SciPy. The amplitude of each band is averaged and then mapped to a colour on an LED strip in real time, so that the LED strip changes colour as the energy of the music changes. The amazing results are sure to be the talk of your party.

The script uses roughly 20% of one CPU core on a Raspberry Pi 2. As a result, it is highly likely that this tutorial will work just fine on any older Raspberry Pi model that may be collecting dust in your drawer. This is an excellent opportunity to get that older model out, blow off the dust and put it to good use.

01 Install dependencies

Start with a fresh Raspbian image and upgrade it to the latest package listings with:

```
sudo apt-get update; sudo apt-get upgrade
```

Then install the required dependencies and next compile PyAudio. PortAudio is used by PyAudio. Pip is used to compile PyAudio as it is not included with Raspbian by default. SciPy is used to filter the audio, while matplotlib is used to plot the frequency response of the filters.

```
sudo apt-get install python-pip python2.7-dev
portaudio19-dev python-scipy python-matplotlib
sudo pip install pyaudio
```

This project also works with line-in, which the Pi doesn't have, so a USB sound card is ideal

02 Disable built-in sound card

We had issues getting the Raspberry Pi's built-in sound card to work reliably for sound output with PyAudio. This project also works with line-in, which the Pi doesn't have, so a USB sound card is the ideal solution. We will disable the built-in card so that the default card is the USB one. To disable the built-in card, you need to do the following:

```
sudo rm /etc/modprobe.d/alsa*
sudo editor /etc/modules
```

Change **snd-bcm2835** to **#snd-bcm2835** and save.

```
sudo reboot
```

03 Test the sound card output

If you're using the Pi to play audio rather than a line-in, then you'll want to test the output. Type **alsamixer** and then make sure the volume is set to a comfortable level. If you are plugging speakers in, then set to 100%. Then type **speaker-test**, this will generate pink noise on the speakers. Press Ctrl+C to exit if you're happy it's working. Sadly, there's no easy way to test a line-in signal. It will be obvious if it's working once the software is written.

04 Construct the circuit

We are using GPIO pins 20, 21 and 16 for red, green and blue, respectively. These pins go through a 220Ω resistor to the first pin of the transistor (base). The second pin (collector) goes to one of the red, green or blue LED-strip pins. The third pin (emitter) goes to ground. The ground from the DC jack needs to be connected to the same ground as the Pi. Connect the 12V from the DC jack to the strip. When connecting the wires, ensure the wires aren't touching. If anything shorts, that would be bad. Ideally, this circuit would go on a stripboard and be soldered to the LED strip.

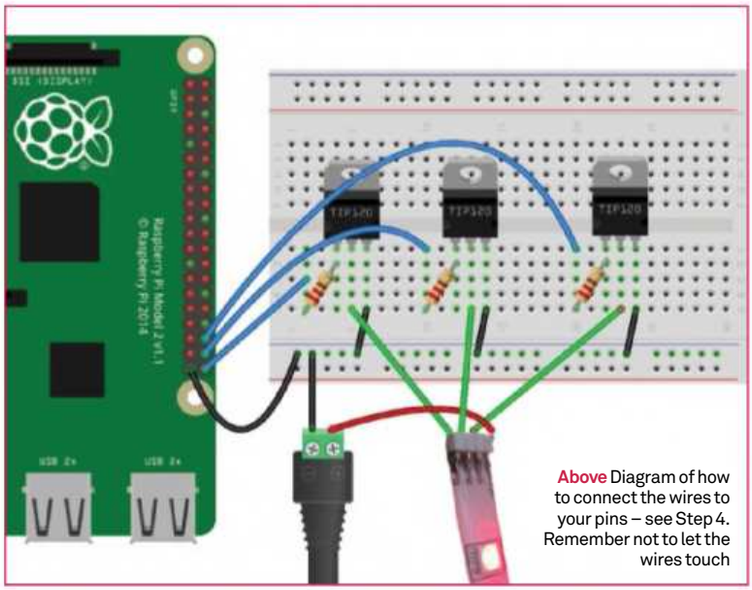
05 Start the script

Create a file called ledstrip.py and mark it as executable. Then begin editing with your favourite editor:

```
touch ledstrip.py
chmod +x ledstrip.py
vim ledstrip.py
```

06 Add imports

After adding the shebang (#!) line, you'll need to add the imports. The GPIO library is used for sending signals to the LED strip. Randrange is used to generate random colours. PyAudio is the audio library. Wave is used to read WAVE files. From SciPy, butter is the filter type that we're using, lfilter is used to apply the filter and freqz gets the frequency response. Matplotlib is used to plot the frequency response of the filters and NumPy is used for fast-math operations on arrays. The back-end of NumPy is written in C, which means it's faster than Python when doing the same operation on a large data set.



Above Diagram of how to connect the wires to your pins – see Step 4. Remember not to let the wires touch

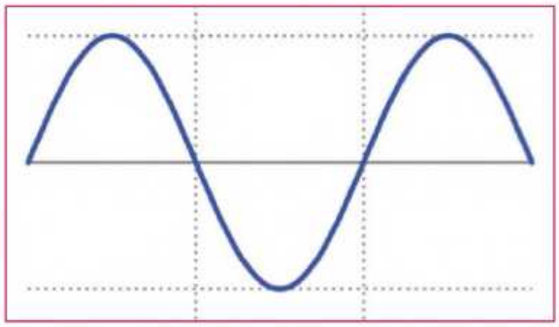
Full code listing

Step 06

```
#!/usr/bin/env python2

import RPi.GPIO as GPIO
from random import randrange
import time
import pyaudio
import wave
from scipy.signal import butter, lfilter, freqz
import matplotlib
#matplotlib.use("GTK") #uncomment when plotting
import matplotlib.pyplot as plt
import numpy as np
import sys

# How many bytes of audio to read at a time
CHUNK = 512
```

Step 07

```
class LedController:
    def __init__(self, pin_nums):
        # pin_nums is an R, G, B tuple

        # Initial setup of GPIO pins
        GPIO.setmode(GPIO.BCM)

        # Set each pin as an output and create a pwm instance
        self.pins = []
        for p in pin_nums:
            GPIO.setup(p, GPIO.OUT)
            # Create a pwm instance for the pin at a
            # frequency of 200Hz
            self.pins.append(GPIO.PWM(p, 200))
            # Set each pin to a random brightness to begin with
            self.pins[-1].start(randrange(0, 100))

    def set_colour(self, colour_tuple):
        # Takes a colour tuple in the form (R, G, B) where the
        # values are from 0 to 255 > 255 is capped

        for i in range(0, 3):
            # Scale 0 to 255 to a percentage
            scaled = int(colour_tuple[i] *
                         (100.0/255.0))
```



Above A perfect sine wave – see Step 12

07 Create LED class

The LED controller class is simple. The **init** function will take a tuple containing (R, G, B) pin numbers and set up the pins. There is a **set_colour** function that takes an (R, G, B) tuple where the values are between 0 and 255. Finally, there is a **test** function that sets a random colour every second. This class could be reused in a different project with no modifications. The class is well commented so doesn't need any further explanation.

08 Create the frequency analyser

The frequency analyser class is responsible for taking the audio data, then filtering it into either bass, middle or treble, and then controlling the LEDs. As arguments, it takes the number of channels in the audio data (assumed to be one or two), the sample rate (usually 44100Hz) and an instance of the LED-controller class.

09 The init method

After storing the input parameters, we calculate the Nyquist frequency, which is half of the sample rate. Nyquist's theorem says that the sample rate of any analogue signal needs to be at least twice the analogue frequency. This is why CDs are sampled at 44.1KHz (human hearing ends at around 20KHz).

Then design two butterworth filters (a filter designed to have a frequency response as flat as possible). The low-pass filter cut-off is 200Hz and the high-pass cut-off is 4000Hz. The parameters are later used to filter with these characteristics.

10 Colours

Finally, we want the colour changes to the strip to be pronounced and noticeable, with reduced flickering. To achieve this, we can store the max value and use it to have colours fall by a certain amount until the energy of the song pushes them back up. The fall list sets the rate for each colour/frequency in the order (R, G, B/bass, mids, treble). The range of values for a colour is between 0 and 255. The LEDs are set at a frequency of Sample Rate/CHUNK. CHUNK is the amount of data to read at once. We set CHUNK to 512, so 44100/512 = 86 times a second. Keep this in mind when setting your fall values, as they depend on what you think looks good and the style of music.

11 The filter function

The filter function is fairly straightforward. An array of samples called data is passed in where the samples are values between -1.0 and 1.0. This is the standard format for signal processing, as opposed to the 16-bit integers used in WAVE files and CDs. The low- and high-pass filters are then applied using the parameters we calculated before in Step 9. We then get the middle band by subtracting the sum of the low and high filter from the original signal.

Full code listing (Cont.)

Step 07

```
            # Ensure we are giving correct values
            if scaled < 0:
                scaled = 0.0
            elif scaled > 100:
                scaled = 100.0

            #print "{0}: {1}".format(i ,scaled)
            self.pins[i].ChangeDutyCycle(scaled)

    def test(self):
        # Change to a random colour
        while True:
            r = randrange(0, 256)
            g = randrange(0, 256)
            b = randrange(0, 256)
            self.set_colour((r, g, b))
            time.sleep(1)
```

Step 09

```
class FreqAnalyser:
    # Filtering based on
    # http://wiki.scipy.org/Cookbook/ButterworthBandpass

    def __init__(self, channels, sample_rate, leds=None):
        self.leds = leds # Not needed if just plotting
        self.channels = channels
        self.sample_rate = sample_rate
        self.nyquist = float(sample_rate) / 2

        # Filter order - higher the order the sharper
        # the curve
        order = 3

        # Cut off frequencies:
        # Low pass filter
        cutoff = 200 / self.nyquist
        # Numerator (b) and denominator (a)
        # polynomials of the filter.
        b, a = butter(order, cutoff, btype='lowpass')
        self.low_b = b
        self.low_a = a

        # High pass filter
        cutoff = 4000 / self.nyquist
        b, a = butter(order, cutoff, btype='highpass')
        self.high_b = b
        self.high_a = a
```

Step 10

```
        # Keep track of max brightness for each colour
        self.max = [0.0, 0.0, 0.0]
        # Make different frequencies fall faster
        # bass needs to be punchy.
        self.fall = [15.0, 2.5, 5.0]
```

Step 11

```
    def filter(self, data):
        # Apply low filter
        self.low_data = lfilter(self.low_b,
                                self.low_a,
                                data)

        # Apply high filter
        self.high_data = lfilter(self.high_b,
                                 self.high_a,
                                 data)

        # Get mid data by doing signal - (low + high)
        self.mid_data = np.subtract(data,
                        np.add(self.low_data,
                               self.high_data))
```

Full code listing (Cont.)

Step 12

```
@staticmethod
def rms(data):
    # Return root mean square of data set
    # (i.e. average amplitude)
    return np.sqrt(np.mean(np.square(data)))
```

Step 13

```
def change_leds(self):
    # Get average amplitude
    l = []
    l.append(self.rms(self.low_data))
    l.append(self.rms(self.mid_data))
    l.append(self.rms(self.high_data))

    # These values are floating point from 0 to 1
    # and our led values go to 255
    divval = 1.0/255

    for i in range(0, 3):
        l[i] = l[i] / divval

    # Do any number fudging to make it look better
    # here - probably want to avoid high values of
    # all because it will be white
    l[0] *= 2 # Emphasise bass
    l[1] /= 2 # Reduce mids
    l[2] *= 5 # Emphasise treble
    #print l

    for i in range(0, 3):
        # First cap all at 255
        if l[i] > 255.0:
            l[i] = 255.0

        # Use new val if > previous max
        if l[i] > self.max[i]:
            self.max[i] = l[i]
        else:
            # Otherwise, decrement max and use that
            # Gives colour falling effect
            self.max[i] -= self.fall[i]
            if self.max[i] < 0:
                self.max[i] = 0
            l[i] = self.max[i]

    self.leds.set_colour(l)
```

Step 14

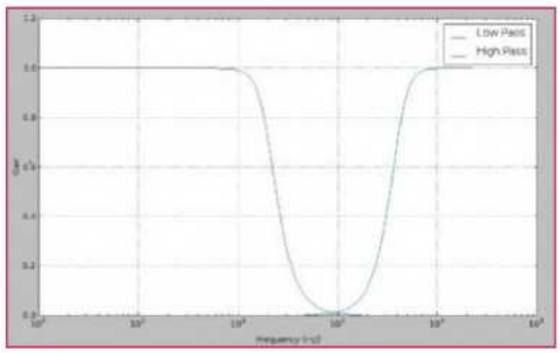
```
def plot_response(self):
    # Frequency response of low and high pass
    # filters. Borrowed from
    # http://wiki.scipy.org/Cookbook/ButterworthBandpass
    plt.figure(1)
    plt.clf()
    w, h = freqz(self.low_b,
                 self.low_a,
                 worN=20000)
    plt.plot((self.nyquist / np.pi) * w,
             abs(h), label="Low Pass")

    w, h = freqz(self.high_b,
                 self.high_a,
                 worN=20000)
    plt.plot((self.nyquist / np.pi) * w,
             abs(h), label="High Pass")

    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Gain')
    plt.grid(True)
    plt.legend(loc='best')
```

12 Root mean square

The root mean square, or quadratic mean, is used to get the average amplitude of the samples in each frequency band. Why are the values squared and rooted? Imagine a perfect sine wave (as shown in the image) that goes between -1 and 1. If you averaged the samples of one cycle using the normal mean method, then the value would be zero because the positive and negative samples would cancel each other out.

13 Change LEDs

Now comes the fun part. The **change_leds** function uses the filtered data, gets the average amplitude of each set and converts it back to a value between 0 and 255. After that, you have the opportunity to fiddle with the values to emphasise frequency values. It is these and the values of the fall array from Step 10 that will determine how your LED strip behaves. When setting a colour, we update the max if the new value is higher than the previous one. If not, we decrement the max by the fall value and use that as the colour to set. This makes the colours fall in a controlled manner when the energy drops. Also, make sure you don't go out of bounds when doing these calculations.



14 Plot frequency response

The **plot_response** function isn't a necessary part of the program, but we used it to draw the frequency response of the filters to ensure they would behave as expected. As such, it is nice to have if you want to have a go at changing the filters. The frequency scale is a log scale because that's how human hearing works. If you log into the Pi with **ssh -X pi@ip.address** and call the function, you should get the plot forwarded to your local machine.

15 The audio controller

The audio controller is the last piece of the puzzle. It is responsible for either playing a WAVE file or capturing a line-in signal, and then sends that data off to be filtered and displayed on the LED strip. The init method takes a filename (or line-in) and an instance of the LED Controller class. A flag is set indicating if the line input is being used. If not, the WAVE file is opened. Finally, the LED instance is stored and an instance of the PyAudio library is created.

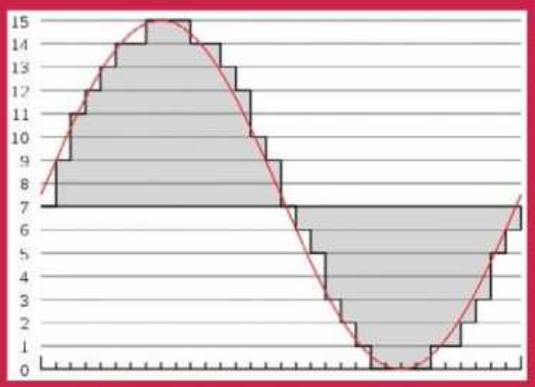
16 Help methods

There are a couple of utility methods in the audio controller. The first one gets the left-hand side of a stereo signal because there's no point analysing both sides. There is also a **more** function, which gets another chunk of audio data from either the WAVE file or line input. Oddly, there is always a line-in error when reading from it the first time. It should stay stable after this – if not, try changing the chunk size. If there is an error, we just return random data.

Audio signal processing

Capturing Audio

Audio is an analogue signal. The first step in processing it is to capture it. To do this, we need to sample the signal. Audio enters the sound card through the line-in port. This signal is fed into an analogue to digital converter, which converts the analogue voltage to a value between -32,768 and +32,767 (if using a 16-bit ADC). The sample rate is the frequency that the signal is sampled. CD-quality audio is sampled at 44.1KHz. The image below shows a sine wave that has been sampled.



Why floating point?

Why do we use a floating point between -1.0 and 1.0 instead of integers when performing signal processing? To quote *Designing Audio Effect Plug-Ins in C++* by Will Pirkle: "In audio algorithms, addition and multiplication are both commonplace. With integer-based numbers, you can get into trouble quickly if you mathematically combine two numbers that result in a value that is outside the range of known numbers... However, numbers between -1.0 and 1.0 have the interesting property that their product is always a number in that range."

The simplest low-pass filter

Signal processing is a very complicated topic so this is supposed to be an explanation of how the simplest possible filter works. Suppose we have a filter y(n) = x(n) + x(n-1), where y is the output sample and x is the input sample, x(n-1) being the previous sample. The low-pass filter takes differing views (shifted in time) of the signal and adds them together.

At low frequencies, all the views look very similar (shifting by a single sample barely changes where you are on the signal at any instant in time). In this case, the two versions will add together in a constructive (or at least non-destructive) way, so the signal passes through the filter.

Now moving to higher frequencies, each shifted version of the signal becomes more distinct at any given instant (sample point) and in fact may even reverse in sign. At these higher frequencies, the different versions of your signal tend to cancel out (added destructively) so the signal now becomes attenuated.

Full code listing (Cont.)

Step 14

```
        plt.xscale('log')
        plt.show()
        # Exit at after showing the plot. Only to
        # verify frequency response
        sys.exit()
```

Step 15

```
class AudioController:
    def __init__(self, filename, leds):
        if filename == 'line-in':
            self.line_in = True
        else:
            self.line_in = False
            self.wf = wave.open(filename)

        self.leds = leds
        self.p = pyaudio.PyAudio()
```

Step 16

```
    @staticmethod
    def get_left(data):
        # Return the left channel of stereo audio
        data = np.reshape(data, (CHUNK, 2))
        return data[:, 0]

    def more(self):
        if self.line_in:
            try:
                # Return line in data
                return self.stream.read(CHUNK)
            except:
                print "line-in error"
                return 'ab'
        else:
            # Read data from wav file
            return self.wf.readframes(CHUNK)
```

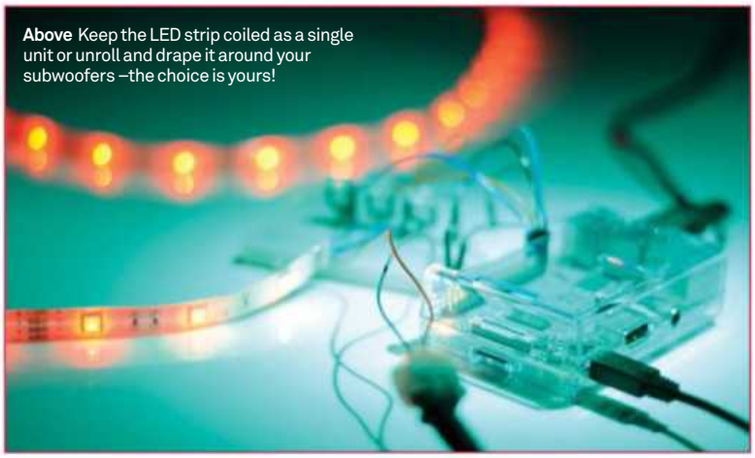
Step 17

```
    def analyse(self, data):
        # Convert to numpy array and filter
        data = np.fromstring(data, dtype=np.int16)

        # If stereo only work on left side
        if self.channels == 2:
            data = self.get_left(data)

        # Convert int16 to float for dsp
        data = np.float32(data/32768.0)

        # Send to filter
        self.analyser.filter(data)

        self.analyser.change_leds()
```

Step 18

```
    def play_setup(self):
        # Assume 16 bit wave file either mono or stereo
        self.channels = self.wf.getnchannels()
        self.sample_rate = self.wf.getframerate()
        self.stream = self.p.open(format = pyaudio.paInt16,
                                  channels = self.channels,
                                  rate = self.sample_rate,
                                  output = True)

    def record_setup(self):
        self.channels = 1
        self.sample_rate = 44100
        self.stream = self.p.open(format = pyaudio.paInt16,
                                  channels = self.channels,
                                  rate = self.sample_rate,
                                  input = True)
```

Above The strip we're using can be bought here: **amzn.to/1Lkutx6**



Above Keep the LED strip coiled as a single unit or unroll and drape it around your subwoofers –the choice is yours!

17 Analyse method

The analyse method is responsible for converting the byte string returned by the more method into an array of 16-bit integers. Then if the audio is stereo, the right-hand side is discarded. The data is then converted into floating-point representation where each sample is between -1.0 and 1.0. This is the standard data format for signal processing. The floating-point data is then sent to the frequency analyser. Finally, the **change_leds** method of the frequency analyser is called.

18 Setup methods

The main function of the audio controller is the loop method. Before the loop begins, a setup helper function is called, which initialises PyAudio to either record from line-in or play back a WAVE file.

19 Loop method

As previously mentioned, the main processing loop begins by setting up the appropriate audio stream. It then initialises the frequency analyser and starts a while loop that runs until there is no audio data left. The audio data is sent to the sound card if we are playing a WAVE file and then sent to the analyser, which deals with the actual analysis and changes the LEDs.

20 The main method

The main method is really simple to perform. The first two lines are for plotting the frequency response of the filters. After this, we create an instance of the LED Controller by passing a tuple containing the GPIO pin numbers for the (red, green, blue) transistors. We then create an instance of the Audio Controller, which passes through both the first command line argument (either line-in or a WAVE file) and also the LED Controller instance we just created. Finally, we enter the processing loop.

21 Start at boot

Now that we have the code finished, it's time to make it start at boot. The application needs to run as root, so we can simply add it to **rc.local**. Edit **/etc/rc.local** with **sudo editor** and add the line:

```
python2 /home/pi/ledstrip.py line-in &
```

... before the `exit 0` line. Now reboot and test your visualiser!

> The analyse method converts the byte string returned by the more method into an array of 16-bit integers

Full code listing (Cont.)

Step 19

```
def loop(self):
    # Main processing loop
    # Do appropriate setup depending on line in or not
    if self.line_in:
        self.record_setup()
    else:
        self.play_setup()

    self.analyser = FreqAnalyser(self.channels,
                                 self.sample_rate,self.leds)

    # Read the first block of audio data
    data = self.more()

    # While there is still audio left
    while data != '':
        try:
            # If we're playing audio write to stream
            if not self.line_in:
                self.stream.write(data)

            # Analyse data and change LEDs
            self.analyse(data)

            # Get more audio data
            data = self.more()
        except KeyboardInterrupt:
            break

    # Tidy up
    self.stream.close()
    self.p.terminate()
```

Step 20

```
if __name__ == "__main__":
    #f = FreqAnalyser(2, 44100)
    #f.plot_response()

    lc = LedController((20, 21, 16))
    #lc.test()
    ac = AudioController(sys.argv[1], lc)
    ac.loop()
```

LED bulbs The ceiling is made up of six main sections containing 2,800 bulbs each, for a total of 16,800 bulbs holding 50,400 RGB LEDs, and thus over 150,000 individual LED components

WizYasep Designed to transmit and transcode data flows through Ethernet connections at high speed, low latency and low jitter, the WizYasep controller board is perfect for networked LED displays

Raspberry Pi Conductor of the light display, the Raspberry Pi processes the preloaded videos and sends them out across the network

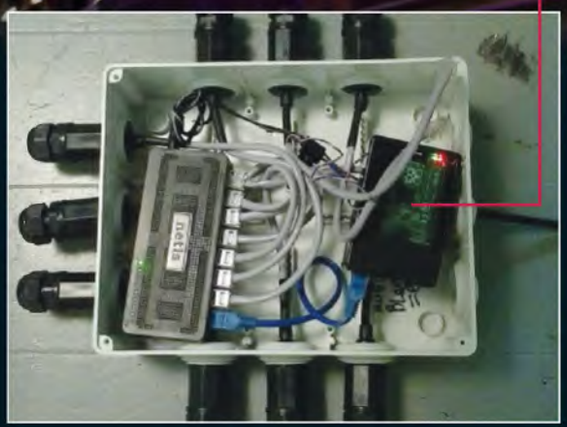
Sensors A pair of sensors, placed at each end of the pathway, are used to calculate when passengers are moving in a particular direction

Components list

- Raspberry Pi B+
- Real-time clock module
- 8-port Ethernet hub
- Raspberry Pi power supply
- 6 x WizYasep boards
- 120 x 150W power supplies
- 50,400 x RGB LEDs (3/bulb)
- 4 x infrared sensors

Right The Pi is in a separate enclosure, transmitting the data out to each of the six WizYasep-managed sections of the ceiling

Below The custom WizYasep board that Yann designed for the project, which is excellent for driving high-volume LED displays



ElectroSuper

Fred Sapey-Triomphe and Yann Guidon make Mons railway station sparkle with a supersized LED installation

The ElectroSuper installation at Mons station looks amazing! How long have you been collaborating?

Fred Yann and I have been working together since February 2013. I wanted to create a large-scale LED display, and I'm a visual artist and have no foundation in electronics, so I couldn't make it by myself. I met Yann through common friends and I went to his studio early in 2013. I was asked to do another project for a show in Normandy, France, so I had a little budget for that and I asked Yann if he would like to start working on that project.

How did you begin work on the ElectroSuper project?

Fred Well, basically we had a show in 2014 in Pompidou Centre, in the east of France, close to the German border in Metz and very close to Belgium. And there is a guy there in Metz who told me that the city of Mons is looking for something to dress up the railway station. [Ed: Mons is a European Capital of Culture this year.] This guy said, "Why don't you send your portfolio to Mons?" So that's what I did and we finally signed the contract three months before the opening. We only had two months to produce the whole piece, which is a 42 metre long ceiling screen.

The screen interacts with passers-by in the tunnel – how does that work?

Fred The idea was to cover the ceiling of a passenger path. People getting off the train have to take this path to the station, so this is the first thing visitors are going to see as they arrive. We were asked to create something engaging, powerful, colourful, something that would put the visitor in a good mood for their visit. We wanted it to be interactive, so Yann put in four infrared sensors, at the entry and exit points of the tunnel. The images that are displayed by the screen are changing according to the number of visitors.

Yann I put each pair of sensors one metre apart, so when I pick up a series of pulses I know that something is moving in one direction on the pathway.

So it's kind of tidal, then – every time there's a new movement, it sends another wave of colour?

Fred Yes, it's a good way to explain it. Also, this project is running for the whole year. We designed it so that the visual effect varies according to the seasons. So right now it's August and the amount of light is larger than in December, so we had to create specific videos for each season.

Aside from the sensors, what other hardware are you using?

Yann Fred designed the whole structure and helped build it. There are a lot of wooden structures with special treatment for the wood because it has to sustain snow, rain and sun. He found premade elements on which we could affix LED strips, 2 by 3.5 metres, like a tile. So we split the whole surface into six sections; each section is 40 by 70 bulbs, so 2,800 bulbs. We have 16,800 in total and it's about one watt per bulb, so if you multiply everything you get more than 16 kwatts. The length is 42 metres and we have to transmit data across this distance. It creates a problem of integrity, reliability, etc, so I chose to use Ethernet for the transmission of data because it's cheap and well supported.

We are very careful about reliability and we have a lot of experience now with making something that is not too expensive, but also that works and sustains the weather and other injuries. Many people will start by driving a WS2812 with Arduinos, which works with one strip, and then to make a screen they will add in more and more. And it will work on the table, but when you move to outdoor installations, the constraints are enormous and Arduino doesn't cut it. So I created a special board, the WizYasep, for driving thousands of LEDs and putting them in parallel to drive even more and make it more reliable.

So you are using a different WizYasep board for each section?

Yann Yes. I used a hierarchical structure: at the top we have a single Raspberry Pi B+, which contains a customised Raspbian. I removed everything to do with X Window, so I saved something like 1 GB, which is even more space for storing the videos. I hardened the operating system so it would be in read-only mode. In these installations, we never know if or when the power supply is removed, so there is the risk of wearing out the SD card if it is storing logs etc. There is also a real-time clock, because we are in a public space and there is the issue of saving energy. When the sun is out, from about 10am to 5pm, the system is turned off. And when the WisYasep boards see that no data is coming, they set everything to black so that it draws less current. The Raspberry Pi is connected with a little 8-port hub, 100 Mbit, so that's okay because one frame is about 50 Kb, and multiplied by 25 frames per second, it's less than 1.5 Mbit per second.

Going back to the display, how are you getting the videos to change and interact with the passengers?

Yann Fred prepares video sequences and then he exports those in a special format, so I can then process them and turn the sequences into files that can be read back. From there, I can modify the program, for example, to apply filters or to speed up or slow down the playback. For now, the system is streamlined, it's smooth, because there is not much processing done on the Pi. It just reads a big block of SD card memory and cuts it into pieces, which are then sent in a special order through the network to the six controller boards, according to a quite simple algorithm that optimises the network congestion. It has a big buffer so it can buffer a lot of data, and then all the screens are updated at the same time by receiving a tiny broadcast packet, so it ensures that the screens are perfectly synchronised.

Like it?

The first project that Fred and Yann worked on together was the Rosace: an electronic persistence of vision mill, similar to a zoetrope or a phenakistoscope. You can check it out here in the first part of the video: **goo.gl/NvGmcJ**

Further reading

Interested in Yann's WizYasep board? There's plenty more information at: **goo.gl/eV1dbB**

What you'll need

- **Raspberry Pi 2**
- **USB sound card** (we used a Behringer UCA202)

Full code
FileSilo.co.uk

Code a simple synthesiser

Learn how to write a simple polyphonic synthesiser (and the theory behind it) using Python and Cython

We are going to take you through the basics of wavetable synthesis theory and use that knowledge to create a real-time synthesiser in Python. At the moment, it is controlled by the computer keyboard, but it could easily be adapted to accept a MIDI keyboard as input.

The Python implementation of such a synthesiser turns out to be too slow for polyphonic sound (ie playing multiple notes at the same time) so we'll use Cython, which compiles Python to C so that you can then compile it to native machine code to improve the performance. The end result is polyphony of three notes, so this is not intended for use as a serious synthesiser. Instead, this tutorial will enable you to become familiar with synthesis concepts in a comfortable language: Python.

Once you're finished, try taking this project further by customising the mapping to better fit your keyboard layout, or tweaking the code to read input from a MIDI keyboard.

01 Install packages

Using the latest Raspbian image, install the required packages with the following commands:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python-pip python2.7-dev
portaudio19-dev
sudo pip install cython pyaudio
```

The final step compiles Cython and PyAudio from source, so you might want to go and do something else while it works its magic.

02 Disable built-in sound card

We had issues getting the Raspberry Pi's built-in sound card to work reliably while developing the synthesis code. For

Cython

Cython is a tool that compiles Python down to the C code that would be used by the interpreter to run the code. This has the advantage that you can optimise some parts of your Python code into pure C code, which is significantly faster. This is achieved by giving C types, such as int, float and char, to Python variables.

Once you have C code it can then be compiled with a C compiler (usually GCC) which can optimise the code even further. A downside to using Cython is that you can't run Cython optimised code with a normal Python interpreter. Cython is a nice compromise because you get a similar simplicity to Python code but higher performance than usual. Cython has a profiler which you can run using:

```
cython -a synth.pyx
```

The profiler outputs a html file which shows where to make optimisations, giving insight into how much overhead using Python introduces. For more details go to **http://cython.org**.

that reason, we are using a USB sound card and will disable the built-in card so that the default card is the USB one:

```
sudo rm /etc/modprobe.d/alsa*
sudo editor /etc/modules
```

Change 'snd-bcm2835' to '#snd-bcm2835' and save, then:

```
sudo reboot
```

03 Test sound card

Now we can test the USB sound card. Type **alsamixer** and then ensure that the volume is set to a comfortable level. If you're plugging speakers in, you'll probably want it set to 100%. Then type **speaker-test**, which will generate some pink noise on the speakers. Press Ctrl+C to exit once you are happy that it's working.

04 Start project

Start by creating a directory for the project. Then download one cycle of a square wave that we will use as a wavetable, like so:

```
mkdir synth
cd synth
wget liamfraser.co.uk/lud/synth/square.wav
```

05 Create compilation script

We need a script that will profile our Python code (resulting in synth.html). Generate a Cython code for it and finally compile the Cython code to a binary with GCC:

```
editor compile.sh:
#!/bin/bash
cython -a synth.pyx
cython --embed synth.pyx
gcc -march=armv7-a -mfpu=neon-vfpv4 -mfloat-
abi=hard -O3 -I /usr/include/python2.7 -o synth.
bin synth.c -lpython2.7 -lpthread
```

(Notice the options that tell the compiler to use the floating point unit.) Make it executable with:

```
chmod +x compile.sh
```

Full code listing

```
#!/usr/bin/python2

import pyaudio
import time
from array import *
from cpython cimport array as c_array
import wave
import threading
import tty, termios, sys
```

Step 07

```
class MIDITable:
    # Generation code from
    # http://www.adambuckley.net/software/beep.c

    def __init__(self):
        self.notes = []
        self.fill_notes()

    def fill_notes(self):
        # Frequency of MIDI note 0 in Hz
        frequency = 8.175799

        # Ratio: 2 to the power 1/12
        ratio = 1.0594631

        for i in range(0, 128):
            self.notes.append(frequency)
            frequency = frequency * ratio

    def get_note(self, n):
        return self.notes[n]
```

Step 08

```
cdef class ADSR:
    cdef float attack, decay, sustain_amplitude
    cdef float release, multiplier
    cdef public char state
    cdef int samples_per_ms, samples_gone

    def __init__(self, sample_rate):
        self.attack = 1.0/100
        self.decay = 1.0/300
        self.sustain_amplitude = 0.7
        self.release = 1.0/50
        self.state = 'A'
        self.multiplier = 0.0
        self.samples_per_ms = int(sample_rate / 1000)
        self.samples_gone = 0

    def next_val(self):
        self.samples_gone += 1
        if self.samples_gone > self.samples_per_ms:
            self.samples_gone = 0
        else:
            return self.multiplier

        if self.state == 'A':
            self.multiplier += self.attack
            if self.multiplier >= 1:
                self.state = 'D'
        elif self.state == 'D':
            self.multiplier -= self.decay
            if self.multiplier <= self.sustain_amplitude:
                self.state = 'S'
        elif self.state == 'R':
            self.multiplier -= self.release

        return self.multiplier
```

Full code listing (Cont.)

Step 09

```
cdef class Note:
    cdef int wavetable_len
    cdef float position, step_size
    cdef c_array.array wavetable
    cdef public float freq
    cdef public object adsr
    cdef public int off

    def __init__(self, wavetable, samplerate, freq):
        # Reference to the wavetable we're using
        self.wavetable = wavetable
        self.wavetable_len = len(wavetable)
        # Frequency in Hz
        self.freq = freq
        # The size we need to step though the wavetable
        # at each sample to get the desired frequency.
        self.step_size = self.wavetable_len * \
                         (freq/float(samplerate))
        # Position in wavetable
        self.position = 0.0
        # ADSR instance
        self.adsr = ADSR(samplerate)
        # Is this note done with
        self.off = 0

    def __repr__(self):
        return ("Note: Frequency = {0}Hz, "
                "Step Size = {1}").format(self.freq,
                                          self.step_size)

    cpdef int next_sample(self):
        # Do the next sample
        cdef int pos_int, p1, p2, interpolated
        cdef int out_sample = 0
        cdef float pos_dec
        cdef float adsr

        adsr = self.adsr.next_val()
        # Need to turn the note off
        # synth will remove on next sample
        if adsr < 0:
            self.off = 1
            return out_sample

        pos_int = int(self.position)
        pos_dec = self.position - pos_int

        # Do linear interpolation
        p1 = self.wavetable[pos_int]
        p2 = 0

        # Wrap around if the first position is at the
        # end of the table
        if pos_int + 1 == self.wavetable_len:
            p2 = self.wavetable[0]
        else:
            p2 = self.wavetable[pos_int+1]

        # Inerpolate between p1 and p2
        interpolated = int(p1 + ((p2 - p1) * pos_dec))
        out_sample += int(interpolated * adsr)

        # Increment step size and wrap around if we've
        # gone over the end of the table
        self.position += self.step_size
        if self.position >= self.wavetable_len:
            self.position -= self.wavetable_len
```

06 Start to code

Our code file is going to be called synth.pyx. This extension tells Cython that it is not plain Python code (and as such, can't be ran in a normal Python interpreter). Create the file with your favourite editor and add the imports.

07 MIDI Table

To synthesise the standard note of a piano, we need a table of MIDI values. MIDI notes range from 0-127. MIDI note 60 is middle C on a piano. The MIDI Table class has a 'get note' function that returns the frequency of a note when you give it a MIDI note number.

Above A visual representation of an Attack, Decay, Sustain, Release curve

08 Attack, Decay, Sustain, Release

The ADSR class applies a volume curve over time to the raw output of an oscillator. It does this by returning a multiplier to the note that is a multiple between 0.0 and 1.0. The version we provide has an attack time of 100 ms, a decay time of 300 ms and a release time of 50 ms. You can try changing these values to see how it affects the sound.

The ADSR class does a lot of maths (44,100 times per second, per note). As such, we want to give types to all of the variables so that the maths can be optimised into a raw C loop where possible, because Python has a massive amount of overhead compared to C. This is what the cdef keyword does. If cdef public is used, then the variable can also be accessed from inside Python as well.

09 Generate notes

The **note** class is the core of our synthesiser. It uses the wavetable to generate waves of a specific frequency. The synthesiser asks the note class for a sample. After generating a sample, the ADSR multiplier is applied and then returned to the synthesiser. The maths of this are explained in the synthesis theory boxout on the opposite page.

The note class does as much maths as the ADSR class, so it is optimised as much as possible using cdef keywords. The cpdef keyword used for the next_sample function means that the function can be called from a non-cdef class. However, the main synth class is much too complicated to give static types to absolutely everything.

10 The audio flow

This synth class is the main class of the application. It has two sample buffers that are the length of the buffer size. While one buffer is being played by the sound card, the other buffer is being filled in a different thread. Once the sound card has played a buffer, the callback function is called. References to the buffers are swapped and the buffer that has just been filled is returned to the audio library.

The smaller the buffer size, the lower the latency. The Raspbian image isn't optimised for real time audio by default so you may have trouble getting small buffer sizes. It also depends on the USB sound card used.

Full code listing (Cont.)

Step 09

```
            return out_sample

    class Synth:
        BUFSIZE = 1024
        SAMPLERATE = 44100

        def __init__(self):
            self.audio = pyaudio.PyAudio()

            # Create output buffers
            self.buf_a = array('h', [0] * Synth.BUFSIZE)
            self.buf_b = array('h', [0] * Synth.BUFSIZE)
            # Oldbuf and curbuf are references to buf_a or
            # buf_b, not copies. We're filling newbuf
            # while playbuf is playing
            self.playbuf = self.buf_b
            self.newbuf = self.buf_a

            self.load_wavetable()
            self.notes = []
            self.notes_on = []

            # The synth loop will run in a separate thread.
            # We will use this condition to notify it when
            # we need more samples
            self.more_samples = threading.Event()
            self.exit = threading.Event()

            # MIDI table of notes -> frequencies
            self.midi_table = MIDITable()

        def stop(self):
            print "Exiting"
            self.exit.set()
            self.stream.stop_stream()
            self.stream.close()

        def stream_init(self):
            self.stream = self.audio.open(
                        format = pyaudio.paInt16,
                        channels = 1,
                        rate = Synth.SAMPLERATE,
                        output = True,
                        frames_per_buffer = Synth.BUFSIZE,
                        stream_callback = self.callback)

        def load_wavetable(self):
            # Load wavetable and assert it is the
            # correct format
            fh = wave.open('square.wav', 'r')
            assert fh.getnchannels() == 1
            assert fh.getframerate() == Synth.SAMPLERATE
            assert fh.getsampwidth() == 2 # aka 16 bit

            # Read the wavedata as a byte string. Then
            # need to convert this into a sample array we
            # can access with indexes
            data = fh.readframes(fh.getnframes())
            # h is a signed short aka int16_t
            self.wavetable = array('h')
            self.wavetable.fromstring(data)

        def swap_buffers(self):
            tmp = self.playbuf
            self.playbuf = self.newbuf
            self.newbuf = tmp
            # Setting the condition makes the synth loop
```

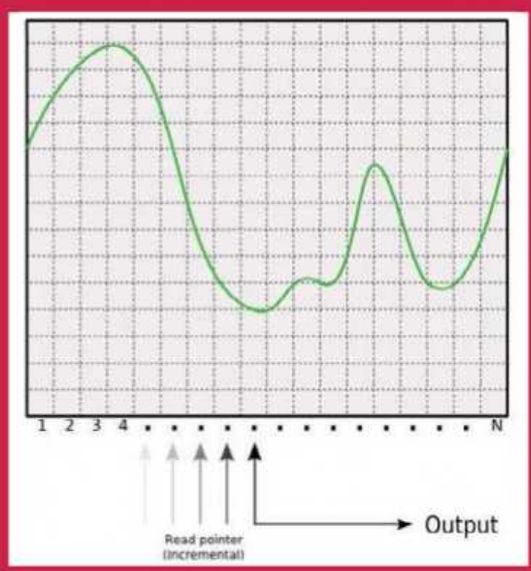
Synthesis theory

Wavetable synthesis is where you use a single cycle of a wave as a lookup table to synthesise sound. In this case we have a square wave, but you can load any wave shape you like. CD-quality audio has a sample rate of 44,100 Hz, which is what we used in our implementation. At each sample, the synthesiser outputs a value from the wavetable and then increments a position pointer to the next value in the table. However, if the wavetable has a frequency of 440 Hz then we need to be able to step through it at arbitrary sizes (ie non-integer values). To achieve this, we use linear interpolation. Assuming the table had a frequency of 440 Hz and we wanted a frequency of 220 Hz, we'd need to step through the table at a step size of 0.5. This can be thought of as drawing a line between two values in the table and picking a value on the line as your output. As an example, if element 0 is 5 and element 1 is 10 then element 0.5 would be 5 + ((10-5) * 0.5), which gives us a value of 7.5. When you reach a position that goes over the end of the table, you wrap around and start again. There is no discontinuity as you're storing a single cycle of the wave in the table. The equation for step size is:

```
step_size = table_size * (note_frequency / sample_rate)
```

The wavetable oscillator gets us a note at the desired frequency, but it's always at maximum amplitude and will sound rough and unnatural. If you cut off a wave in the middle of a cycle there will be a pop or click, so this is where Attack, Decay, Sustain and Release envelopes help. These change the amplitude of the raw oscillator output over time to sound more like an instrument. This is done by applying a fractional multiplier to the original sample point returned by the wave table oscillator. Having a release time from 100% volume to 0% means that a note will fade out smoothly when it's turned off. With the right ADSR curves and the correct wavetable, a synthesiser can sound very similar to real instruments.

More information can be found at: bit.ly/1Kgl9dp.



Above Here's one cycle of a wavetable oscillator

Performance issues

Python introduces a number of performance issues compared to a native synthesiser implementation that is written in C or C++. Cython has been used in our implementation to try and mitigate these issues but it is nowhere near enough. As a rough comparison, our expert worked on a synthesis project targeting 100 Mhz ARM processors that were programmed in C and could get around 30 notes of polyphony, compared to three in this implementation on a 900 Mhz ARM core.

A major issue is that the sound card uses 16-bit signed integers to represent a sample. However, Python doesn't natively support this type. To pass the data to the audio library it needs to be encoded from an array of integers into a byte string. Then at the other end, the Python that talks to the audio library will decode this byte string back into an integer array. If it was written in C or another lower-level language like C++ or Rust, the sample could be passed almost directly to the audio hardware.

Another issue is that Python has a large function call overhead. In compiled languages, this can be optimised out by compiling function calls in line with the caller (effectively, copying the code from the function into the caller). Variable access also has overhead because of all the type checking required. There is also the overhead of the garbage collector, which destroys objects when there are no longer references to them.

A major issue is that the sound card uses 16-bit signed integers to represent a sample. However, Python doesn't support this type

Full code listing (Cont.)

```
        # generate more samples
        self.more_samples.set()

    def callback(self, in_data, frame_count,
                 time_info, status):
        # Audio card needs more samples so swap the
        # buffers so we generate more samples and play
        # back the play buffer we've just been filling
        self.swap_buffers()
        return (self.playbuf.tostring(),
                pyaudio.paContinue)

Step 11
    def do_sample(self, int i):
        cdef int out_sample = 0
        # Go through each note and let it add to the
        # overall sample
        for note in self.notes:
            if note.off:
                self.notes.remove(note)
            else:
                out_sample += note.next_sample() >> 3

        self.newbuf[i] = out_sample

    def synth_loop(self):
        cdef int i

        while self.exit.is_set() == False:
            # For each sample we need to generate
            for i in range(0, Synth.BUFSIZE):
                self.do_sample(i)

            # Wait to be notified to create more
            # samples
            self.more_samples.clear()
            self.more_samples.wait()

    def start(self):
        self.stream_init()
        # Start synth loop thread
        t = threading.Thread(target=self.synth_loop)
        t.start()

Step 12
    def freq_on(self, float freq):
        n = Note(self.wavetable, Synth.SAMPLERATE,
                 freq)
        print n
        self.notes.append(n)

    def freq_off(self, float freq):
        # Set the ADSR state to release
        for n in self.notes:
            if n.freq == freq:
                n.adsr.state = ord('R')

    def note_on(self, n):
        self.freq_on(self.midi_table.get_note(n))
        self.notes_on.append(n)
```

11 Synth loop

The start method of the synth class initialises the audio hardware and then starts the synth_loop method in its own thread. While the exit event is set to false, the do_sample function is called.

The do_sample function loops through the notes that are currently turned on and asks for a sample from each one. These samples are shifted right by three (ie divided by 2^3) and added to out_sample. The division ensures that the output sample can't overflow (this is a very primitive method of adding notes together, but it works nonetheless).

The resulting sample is then put in the sample buffer. Once the buffer is full, the more_samples condition is cleared and the synth_loop thread waits to be notified that the buffer it has just built has been sent to the audio card. At this point, the synth can fill up the buffer that has just finished playing and the cycle continues.

12 Turn on notes

There are both note_on/off and freq_on/off functions that enable either MIDI notes or arbitrary frequencies to be turned on easily. Added to this, there is also a toggle note function which keeps track of MIDI notes that are on and turns them off if they are already on. The toggle note method is used specifically for keyboard input.

13 Add keyboard input

For keyboard input, we needed the ability to get a single character press from the screen. Python's usual input code needs entering before returning to the program. Our code for this is inspired by: **https://code.activestate.com/recipes/577977-get-single-keypress**.

There is a mapping of letters on a keyboard to MIDI note numbers for an entire keyboard octave. We have tried to match the letter spacing to how a piano is laid out to make things easier. However, more innovative methods of input are left as an exercise to the reader.

14 Put it all together

The main function of the program creates an instance of the synth class and then starts the audio stream and synth loop thread. The start function will then return control to the main thread again.

At this point we create an instance of the KB input class and enter a loop that gets characters and toggles the corresponding MIDI note on or off. If the user presses the Q key, that will stop the synth and end the input loop. The program will then exit.

15 Compile the code

Exit your editor and run the compile script by typing the following command:

```
./compile.sh
```

This may take around 30 seconds, so don't worry if it isn't instant. Once the compilation has finished, execute the synth.bin command using:

```
./synth.bin
```

Pressing keys from A all the way up to K on the keyboard will emulate the white keys on the piano. If you press a key again the note will go off successfully. ■

Full code listing (Cont.)

Step 12

```
    def note_off(self, n):
        self.freq_off(self.midi_table.get_note(n))
        self.notes_on.remove(n)

    def toggle_note(self, n):
        if n in self.notes_on:
            print "note {0} off".format(n)
            self.note_off(n)
        else:
            print "note {0} on".format(n)
            self.note_on(n)
```

Step 13

```
class KBInput:
    def __init__(self, synth):
        self.synth = synth

        self.keymap = {'a' : 60, 'w' : 61, 's' : 62,
                       'e' : 63, 'd' : 64, 'f' : 65,
                       't' : 66, 'g' : 67, 'y' : 68,
                       'h' : 69, 'u' : 70, 'j' : 71,
                       'k': 72}
        self.notes_on = []

    @staticmethod
    def getch():
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(fd)
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN,
                              old_settings)
        return ch

    def loop(self):
        while True:
            c = self.getch()

            if c == 'q':
                self.synth.stop()
                return

            if c in self.keymap:
                n = self.keymap[c]
                self.synth.toggle_note(n)

if __name__ == "__main__":
    s = Synth()
    s.start()
    kb = KBInput(s)
    kb.loop()
```

```
note 60 on
Note: Frequency = 261.625640869Hz, Step Size = 0.599187970161
note 60 off
note 64 on
Note: Frequency = 329.627685547Hz, Step Size = 0.754929602146
note 64 off
note 65 on
Note: Frequency = 349.228363037Hz, Step Size = 0.799820065498
note 65 off
note 67 on
Note: Frequency = 391.995574951Hz, Step Size = 0.897767663002
note 67 off
Exiting
```

Above The simple user interface. Notice how the step size in the wavetable varies with frequency

Build a radio transmitter

Take advantage of the interference-blocking feature and make your mark on the airwaves

Back in the 1960s, offshore boats were used to broadcast what was then known as 'pirate' radio: unlicensed broadcasts that provided an alternative to the BBC's light program (as the most populist radio station was then known). Pirate radio was a revolution that inspired Radio 1 and commercial broadcasting, but these days you don't need a boat to pursue your radio DJ dream – just a Raspberry Pi.

Add a basic DIY antenna, an SD card with some MP3 tunes saved to it, plus a script to automate playback, and you can follow in the footsteps of John Peel and Tony Blackburn.

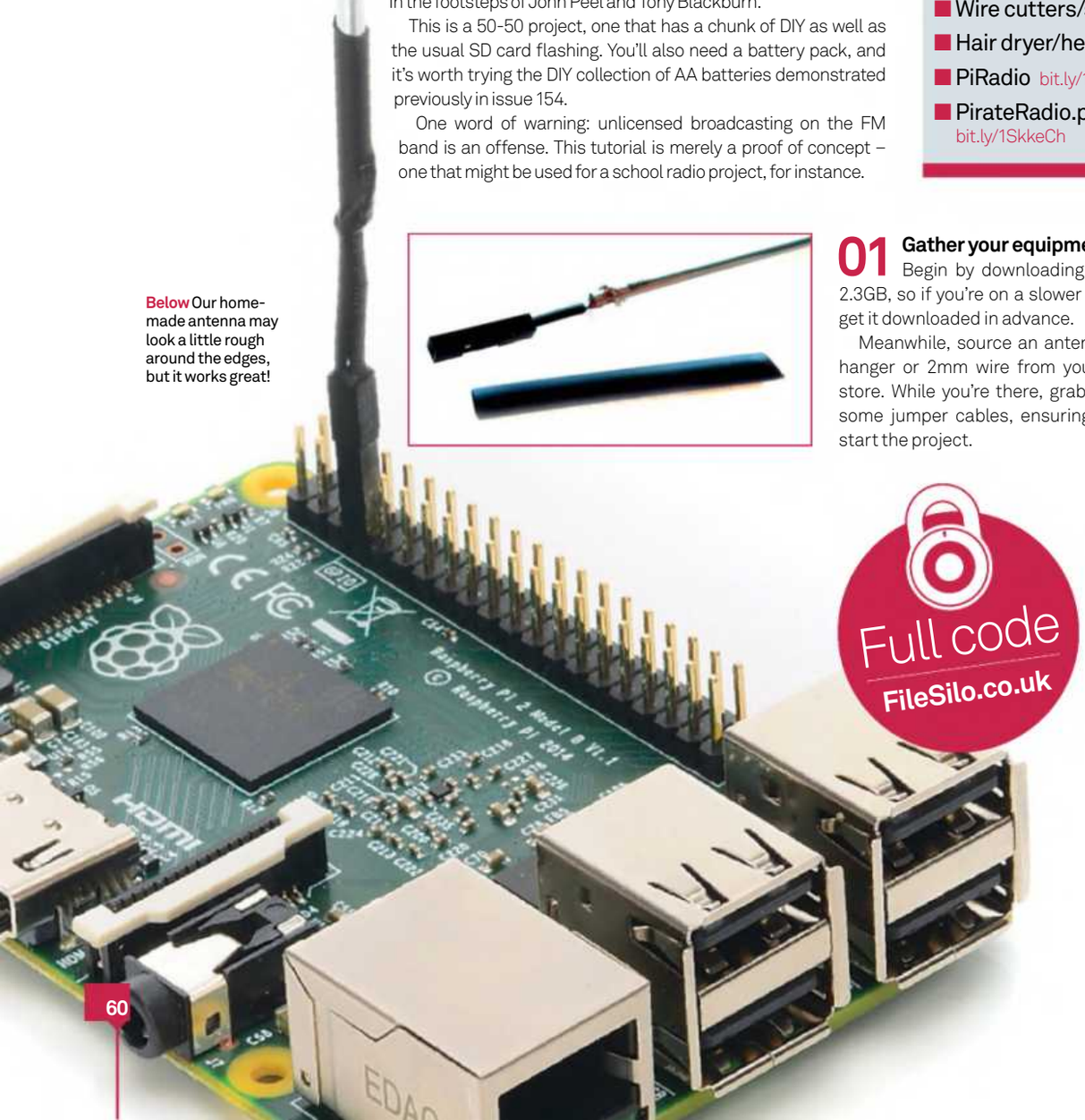
This is a 50-50 project, one that has a chunk of DIY as well as the usual SD card flashing. You'll also need a battery pack, and it's worth trying the DIY collection of AA batteries demonstrated previously in issue 154.

One word of warning: unlicensed broadcasting on the FM band is an offense. This tutorial is merely a proof of concept – one that might be used for a school radio project, for instance.

What you'll need

- Jumper wire
- 2mm wire
- Heat shrink tubing
- Soldering iron
- Wire cutters/strippers
- Hair dryer/heat gun
- PiRadio bit.ly/1MWkxwp
- PirateRadio.py script bit.ly/1SkkeCh

Below Our home-made antenna may look a little rough around the edges, but it works great!



01 Gather your equipment

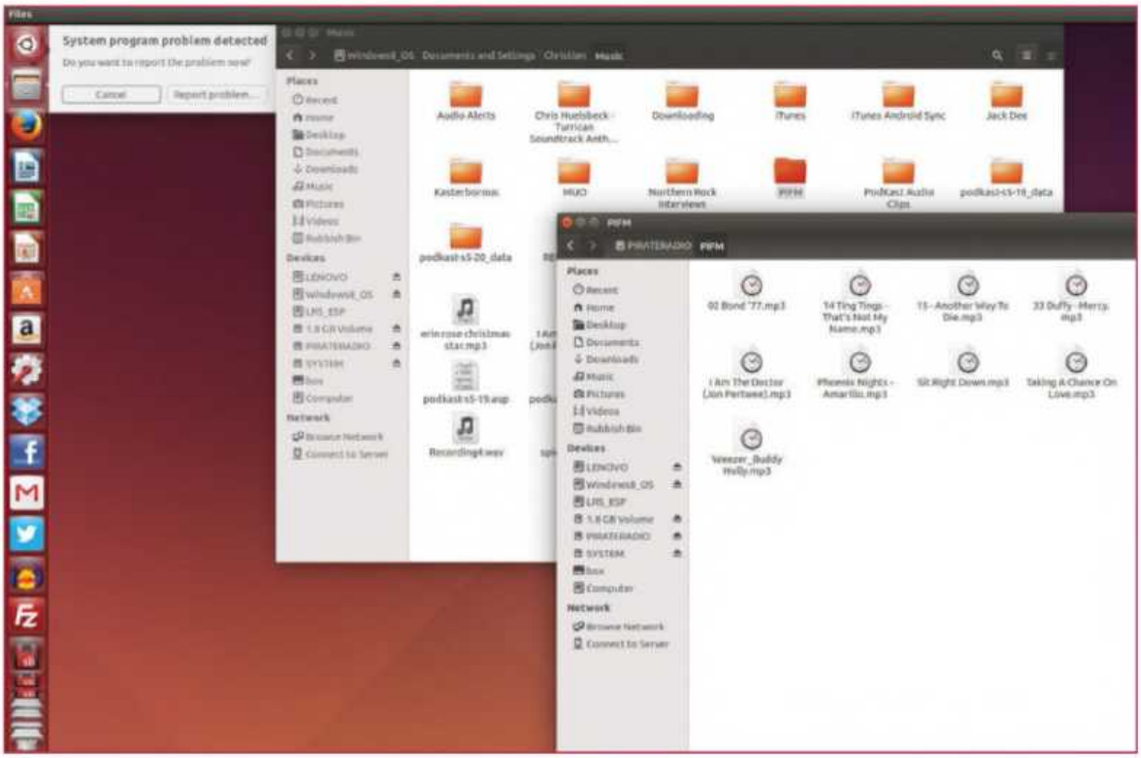
Begin by downloading the PiFM image. This is over 2.3GB, so if you're on a slower connection then you'll need to get it downloaded in advance.

Meanwhile, source an antenna. This might be a wire coat hanger or 2mm wire from your local electronic component store. While you're there, grab some heat shrink tubing and some jumper cables, ensuring that you're well prepared to start the project.

Full code FileSilo.co.uk

Build a case

Okay, so you've already got a suitable case for your Raspberry Pi, but why not go all-out and put together a new case for this project? One idea is to take inspiration from the broadcast motif and design an old-style antenna case, with the Pi and the genuine antenna cleverly hidden inside it. Alternatively, a Mason jar (or other suitably wide-necked jar) will also make a great home for the PiFM – just drill a hole in the lid for the antenna!



Left Prepare MP3 files in advance or make your own using Audacity, or a similar open source audio production tool

02 Prepare MP3 files in advance

There's no facility for live broadcast. Instead, you will need to arrange your MP3 files in advance. These will be played using a script and transmitted to the nearest radio.

Planning on some DJ-ing? All is not lost, as you can produce your own MP3 files using Audacity or similar open source audio production tools. To get the files to play in a particular order, number them, or the folders, sequentially.

03 Power your Raspberry Pi radio

Planning to try this out in a remote area? You'll need a battery pack to power your Raspberry Pi, or else run it from your car's cigarette lighter. For flexibility, you may also need a mobile device with SSH software to wirelessly connect to your Raspberry Pi pirate radio in headless mode.

Of course, the project doesn't have to be portable and you can power your Raspberry Pi as usual.

04 Prepare your broadcast antenna

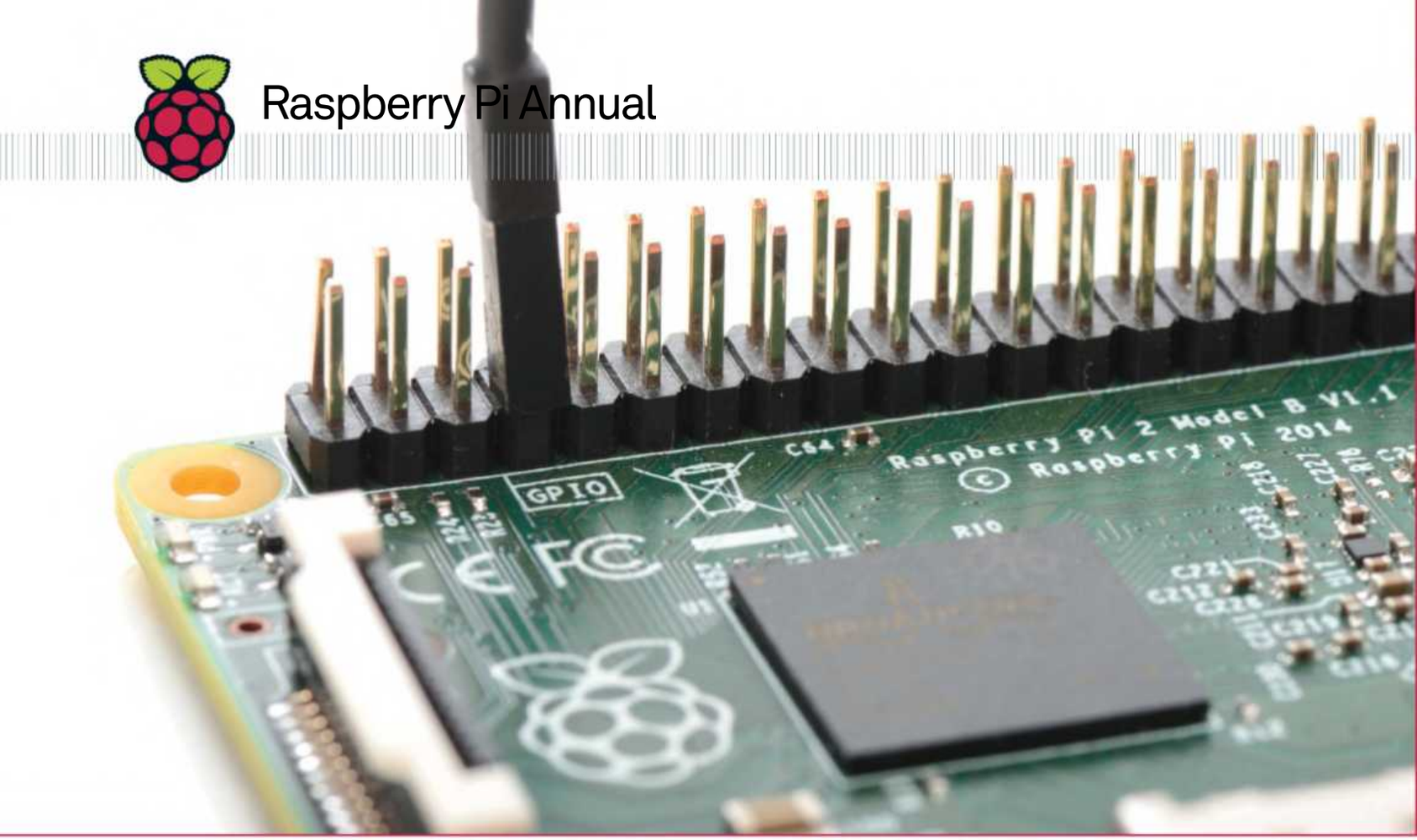
To broadcast from your Raspberry Pi, you'll need a suitable antenna. Electronic retailers stock copper wire that's around 2mm in diameter, but this is usually only available in bulk. As you only need a single 200-250mm length, the best option is to use a handheld rotary blade (or hacksaw) to cut the length from a wire coat hanger.

> Unlicensed broadcasting on the FM band is an offence. This tutorial is merely a proof of concept – one that might be used for a school radio project, for instance

05 Connect the jumper

Strip the wire from a female jumper, leaving enough to solder the 2mm wire to.

Once cooled, add a 50mm length of heat shrink tubing to the top of the jumper and the lower portion of the antenna to insulate the connection. This will tighten as you apply heat from a hairdryer or heat gun. Be careful when heating, as the antenna will warm up and can burn your fingers.



Above The original GPIO 4 hack was discovered by two students at Imperial College London

06 Connect the antenna

The antenna is connected to pin 4 on your Raspberry Pi's GPIO. Before you do this, check that your Raspberry Pi case is suitable for high profile GPIO connections. If not, consider connecting it to a short length of wire and mounting it on top of your case with an adhesive like Sugru, or perhaps just tape it to the side of your case. As long as the antenna has a connection to pin 4, you're good to go.

07 Prepare your SD card

As with all Raspberry Pi projects, it's good to start with a freshly flashed SD card. To get started quickly, use the disk image linked in the kit list, extract the ISO file and flash. However, if you would rather spend some time tweaking the script, flash Raspbian Wheezy and install PirateRadio.py from GitHub.

08 Copy your MP3 files to SD

You cannot simply dump your MP3 files on the SD card. With your flashed SD card still inserted into your PC card reader, browse to the /Pirate Radio partition of the card and paste your copied files.

Beyond simply pasting in numbered MP3s, you can also drop in named folders from your music collection containing entire albums or artist catalogues.

Multiple audio formats

Throughout the tutorial, we've talked about audio files as MP3s, but one of the many beauties of the PiFM project is that it supports other formats. These are re-encoded as required in time for broadcast, based on a playlist created when the Python code scans the SD card for audio files. In addition to MP3s, you can cue up and broadcast files in FLAC, WAV, M4A, AAC and WMA.

09 Configure the Pirate Radio

With the SD card still inserted into your PC, open the pirateradio.config file in a text editor. Look for the frequency setting and adjust this as necessary. You'll need to set this to a frequency that is currently unoccupied, so switch on your FM radio, find some free space and change the pirateradio.config file as necessary. Save and exit the file when you're done.

10 Random and continuous music

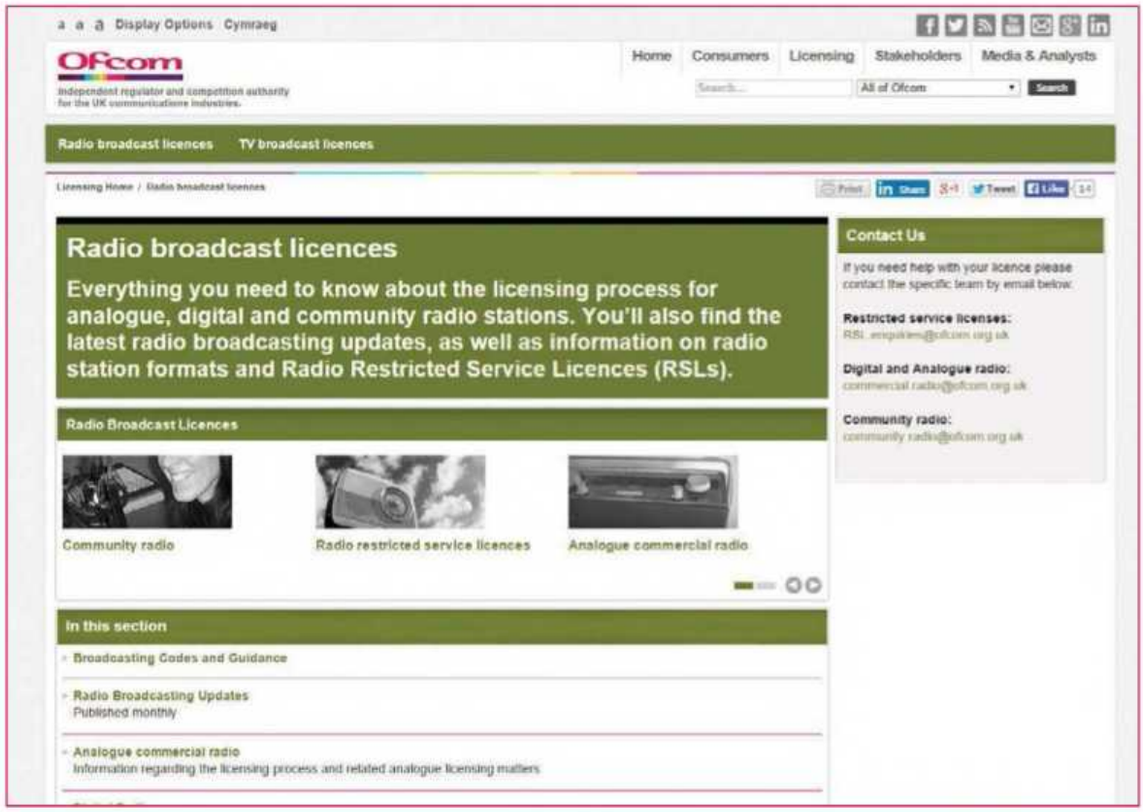
Should you plan to add a lot of music to your SD card for playback on your pirate radio project, you may want to use the **shuffle** and **repeat_all** settings in the pirateradio.config file. By default these are set to true, but to disable, you simply need to change true to false.

Save when you're done, and remember to unmount the SD card before removing it from your computer.

11 Stereo or mono?

The pirateradio.config file offers you the choice of setting a true or false value to the **stereo_playback** value. You should consider this carefully, as it will determine quality and range for your broadcast.

Set to true, the broadcast will be of superior audio quality. However, the range will be reduced as additional power is



How does the Pi broadcast?

Spread-spectrum clock signals on the GPIO pins are the secret power behind the Raspberry Pi's surprising hidden ability to broadcast on the FM band. By utilising this energy with an antenna on pin 4, you can turn a method employed to reduce electrical interference with other devices connected to and situated near your Raspberry Pi into a tool for radio communication.

Left Check out Ofcom's radio broadcasting licenses for more details on the law regarding broadcasting music

required. So setting **stereo_playback** to false will increase your broadcast's range. However, due to the scale of the transmitter, this is only a matter of 20 feet!

12 Appreciate the law

It's all too easy to get into trouble with this project, so before you start broadcasting make sure you're familiar with the law and licensing requirement for broadcasting on the FM band. This project is perfect for short range use, such as playing your MP3s on an old car radio, although you'll have to modify the length of the antenna for this. Schools may also benefit from a Raspberry Pi radio station.

13 Take to the airwaves!

Insert your SD card into the Raspberry Pi and plug the device in. Meanwhile, step away from the device and head into the next room with your FM radio and tune into the specified frequency. Once the Raspberry Pi has booted you should find that the MP3 files are being broadcast!

14 Troubleshooting bad broadcasts

Problems with your broadcast? Check the FM radio is capable of picking up other stations. If you're in the UK, look for Radio 2 on 88-91 FM, which can be received virtually anywhere.

You should also check the frequency setting in the pirateradio.config file. Remember that the band is accessible in the UK from 87.5-108 FM. As such, you cannot access frequencies beyond these points on an FM radio.

15 Check your antenna for problems

Reception issues may be traced back to the antenna. Double-check the soldering is secure and confirm that the wire is connected to GPIO 4. Using the wrong pin won't harm your Raspberry Pi, but equally it won't result in audio being broadcast to your FM radio!

> This project is perfect for short range use, such as playing your MP3s on an old car radio, although you'll have to modify the length of the antenna for this

Interference in the broadcast may be due to wireless routers and microwave ovens. Your Raspberry Pi's power source may also cause problems.

16 Elevate your broadcast

You can improve the range by positioning your Raspberry Pi and the PiFM antenna in an elevated position. You might, for instance, place it by an upstairs window.

Even better results can be achieved by broadcasting from up a tree or on a hillside – but keep in mind that an unlicensed project really needs to maintain a short range.

17 Curb your piracy

You've built a compact, potentially portable radio transmitter, but remember that it isn't the 1960s and you're not a revolutionary. Consider the approved uses for the project and stick to these, rather than using it to cause mischief and get a criminal record. You could make an FM repeater, for example, or create a wireless mic. If you really want to get on the radio, consider volunteering with your local community station.

Stream media through your network with OSMC

Set up a Samba server and map a network drive to OSMC to stream media across your home network

Thousands of images, videos and music files live in your pocket and it's likely even more are in your personal cloud or hard drives. While it's great enjoying these on our personal devices, it's gratifying to see them on our big TV screens – but this isn't easy. Do we hook up our laptops? Do we fiddle with a menagerie of wires and resolutions, hoping for a decent image? Will the sound work? All of this is taken care of with a Raspberry Pi 2 and OSMC. OSMC is an OS built to do one thing: put your media up on that big screen. But, how do we get media onto OSMC? Here we need Samba and a networked hard drive.

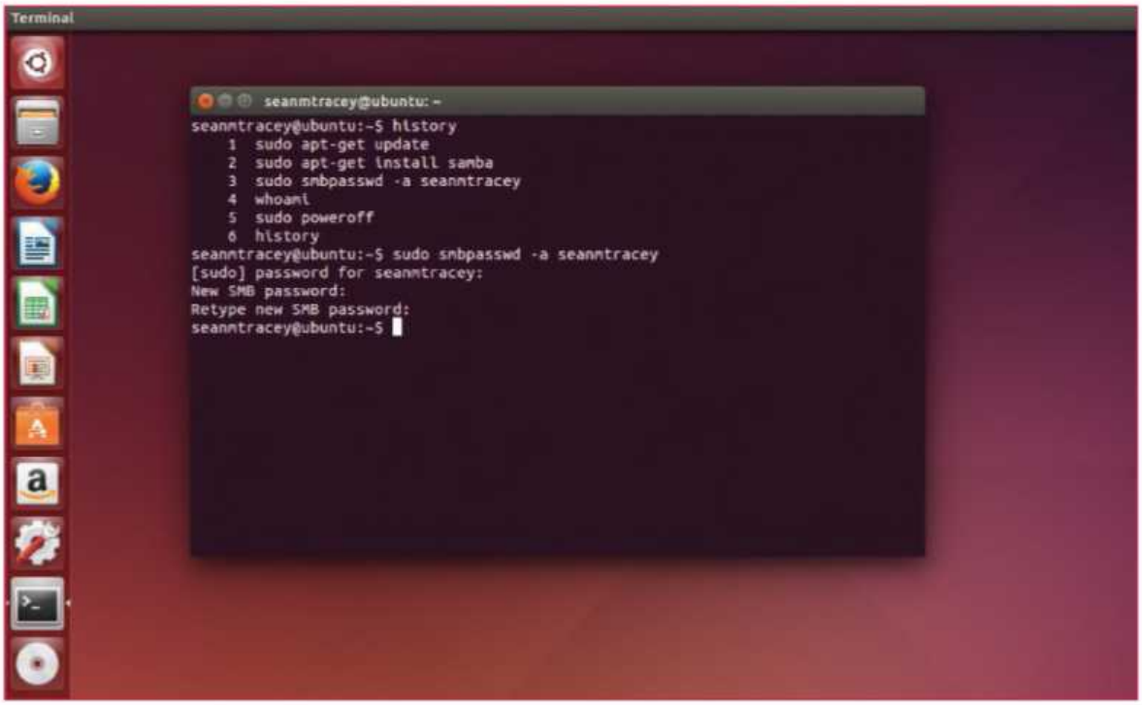
01 Install Samba

We're going to assume that you already have a networked OSMC Pi set up already (it's a simple installation), so we'll jump straight into setting up Samba. For this tutorial, we're using Ubuntu 14.04. If you're running a Debian system, all of these instructions should work fine as they are. If not, you may have to search around for the appropriate repos. First, we need to get the Samba package from the repository with **apt-get**. Open up a terminal and enter:

```
sudo apt-get install samba
```

What you'll need

- OSMC osmc.tv
- Hard drive
- Home network
- Samba
- Another Linux computer, less than eight years old



Left If you run **smbpasswd** as root, you can also add and remove users from the smbpsswd file

02 Add a new user to Samba

Once installation has completed, our system now has the ability to share folders and drives with the Samba protocol. Before we can start mapping paths on our home network, we need to create a password. For brevity's sake, we'll use the same user that we're logged in as to manage Samba. Once you enter the following command, you'll be prompted to enter a password.

```
sudo smbpasswd -a YOUR_USER_NAME
```

03 Prepare hard drive on the network

In order to mount our volumes for sharing, we need to add some configuration settings to the samba.conf file. *.conf files are quite an important deal, so let's make a copy of ours just in case something goes wrong and we find ourselves tearing out our hair trying to undo the damage.

```
sudo cp /etc/samba/smb.conf ~/Desktop/
```

04 Plug in your drive

If you've had the hard drive you're intending to share plugged-in all this time, don't panic, you've done nothing wrong, but if you haven't then now is the time to hook it up. Open up your file manager window, right click on the drive's icon and click Properties. Take note of the path and volume name.

05 Configure Samba

Now that we have a backup of our Samba configuration and we know the path of our media drive, we can set up sharing on our network. Using a command line editor of your choosing (one of our favourites is Vim), open **/etc/samba/smb.conf** for editing and go to the end of your file:

```
> sudo vi /etc/samba/smb.conf
```

```
[DRIVE_NAME]
path = path/to/drive/drive_name
available = yes
valid users = user_you_entered_earlier_for_samba
read only = no
browseable = yes
public = yes
```

Exit and save. Now restart Samba with:

```
> sudo service smbd restart
```

Assuming everything went well, you now have a Samba drive set up for sharing on your home network – so long as it's plugged in.

06 Get the address of the drive

Before we jump over to OSMC, we want to find out the address of our Samba drive. Staying in your terminal, enter **ifconfig** and you'll get a readout of all of your network interfaces:

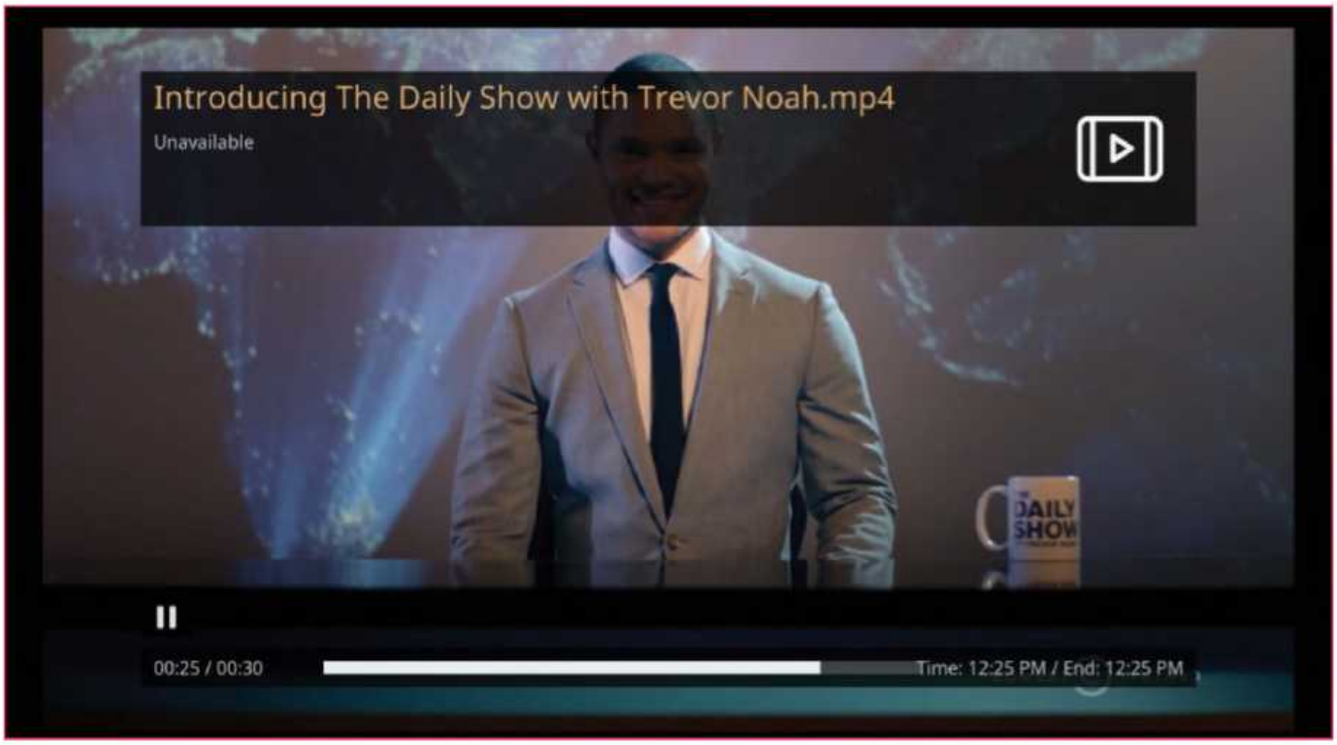
```
eth0    Link encap:Ethernet HWaddr 08:00:27:14:6b:6e
  inet addr:192.168.1.5 Bcast:192.168.1.255 Mask:255.255.255.0
  inet6 addr: fe80::a00:27ff:fe14:6b6e/64 Scope:Link
  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
  RX packets:3874 errors:0 dropped:0 overruns:0 frame:0
  TX packets:5394 errors:0 dropped:0 overruns:0 carrier:0
  collisions:0 txqueuelen:1000
  RX bytes:737927 (737.9 KB) TX bytes:11086898 (11.0 MB)
```

You won't get the exact same output, but you should have something like the second line buried somewhere. Look for **inet addr:** and copy down the IP – this is the address we'll use to access our Samba share over the network. Remember, we're using the Samba protocol, so the address will be (in our case) **smb://192.168.1.5/MEDIADRIVE**. A Samba address is made up of the protocol, IP and name that we gave our Samba share.

```
smb://  192.168.1.5/ MEDIADRIVE
[PROTOCOL]  [ADDRESSS] [SHARE NAME]
```

Why Samba?

When it comes to network protocols, we're truly spoilt for choice. But why Samba over, say, HTTP or an FTP server to deliver files? Well, in order to get things with HTTP or FTP and so on, you often need to know how to get to them beforehand. That makes complete sense in a server-client world, but Samba has discoverability built-in. Why spend time tweaking URLs and configurations when we can have our devices tell us how to find them?



Above Don't have an OSMC remote? Try the new Kore remote app from the Kodi devs: **bit.ly/1JPNs2L**

Pi & power

The Pi, even the new version 2, is not renowned for its blistering speed. So how come it can handle streaming and display high-definition content with ease, whereas it might struggle to run a modern version of Chrome and WebGL? Every Raspberry Pi has a hardware H.264 decoder built-in. This specialised chip does one thing really well, really quickly: it plays H.264 video super-fast. That's the ace up the Pi's sleeve that makes it one of the cheapest and best options for home-made media centres like this.

07 Add our drive to OSMC

Right, time to leave your Linux box alone – let's do some Linux! Hop on over to your OSMC system. If you're one of the lucky people with the super-cool OSMC remote, you can do everything you need to do with it, but there's quite a bit of typing ahead, so if you have a keyboard lying around then now is the time to plug it in. Power up your system and go to Videos>Files>Add Videos. You'll be presented with a view like the picture above.

Enter the address we jotted down in the previous step – in our case it would be **smb://192.168.1.5/MEDIADRIVE** – and hit Done.

08 Appease the password gods

Depending on your setup, you may or may not be asked to enter a username and password. If you do, you'll be prompted as soon as you've entered the path to your media drive – but what is it? That's simple: it's just your ordinary username and password for accessing your system. Once you've entered those, tick the 'Remember this path' box and OSMC will keep a track of the drive and its content as long as it's available on the network.

09 Test the setup

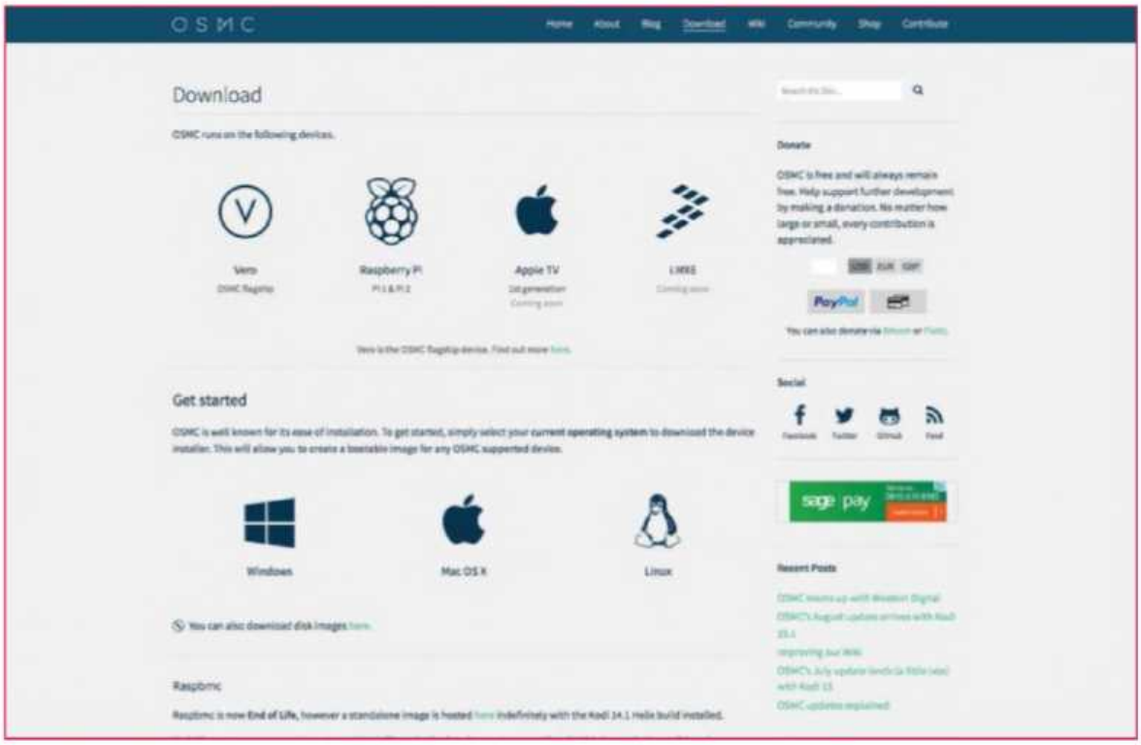
In your videos screen you should now see your shared network drive, which looks just like any other folder. Now that the drive has been mapped, it will reflect any and all changes, structures and files that you apply to your network drive. Select a video to see if it plays. If it does, congratulations! You've now got a networked Samba drive to deliver all of your media.

10 Media types and the Raspberry Pi

You may have noticed, just before you tested your video, a bunch of small black boxes with icons and numbers inside them, just to the left of the media selector UI. These display the properties of video files that OSMC is familiar and comfortable with. Don't worry, if you have videos in a slightly unconventional format (Matroska or WebM, for example) OSMC has a ton of goodies built-in to handle things it's unsure of – it might just take a little while for it to figure it out.

11 Add extra media from elsewhere

So, we now have OSMC reading media files from our Samba-serving PC. But what if we want to add more media? Well, OSMC is geared towards watching your media rather than managing it. We could unplug the media drive from our Samba server, plug it into another computer with the media file we want and then plug it back in to our samba server. Samba can handle plugging drives in and out, but that's cumbersome. Alternatively, we could log in to our Samba server and move files over to it with SCP or FTP, but that's not very user-friendly. What if we could drag media files over the network onto our drive so that we could view them on OSMC?



Left To find the Pi SD card images, click the text link just below the three OS logos shown, or go straight to **osmc.tv/download/images**

12 One step forward, seven steps back

Back in Step 5, we set up our MEDIADRIVE Samba share by editing the Samba config file and we have forgotten all about it since. Well, there's more we can do with Samba. As well as making drives readable on our network, we can also make them writable, so let's do that. Open up /etc/samba/smb.conf with your favourite editor and find your drive's Samba configuration (this should be at the very bottom of the file). Right at the end, just after **public=yes**, add:

```
writable = yes
```

Exit and save.

13 So did that work?

Now we should have a working Samba service running in the background of your computer, but first we need to restart it to make our changes take effect. Before we do that, though, we can check which properties will take effect on restart with the command **testparm**. Run it as an ordinary user (you shouldn't need to **sudo** here), press Enter when prompted and look at the output. If you aren't shown any error messages, your config should be fine. So, it is the time to restart Samba again with **sudo service smbd restart**.

14 Drag, drop, play and enjoy

We now have a network-visible and writable hard drive that OSMC can play files from. You can drag the personal files the you want to watch from one computer on your network onto the Samba drive and have them pop up on OSMC straight away. Let's imagine you use OS X in your day-to-day life (sacrilege!). To add a file to OSMC, you would only have to open Finder, click Go>Connect to Server, then enter the Samba address for your drive. Next, enter the username and password, just like before, and then drag the file into the window. Now that file is on OSMC and you didn't even have to touch it!

> Remember that you don't have to share your entire drive, only a certain folder with your drive

15 More please

If you want to add multiple drives to OSMC, you can repeat Steps 4 through 9. But remember that you don't have to share your entire drive, only a certain folder within your drive – all you have to do is change the path variable to point at a folder. The address will still be the same because it's made up of the IP address and Samba name of your drive. When used properly, this can help you to manage your media over the network.

16 When the party is over

If you find yourself wanting to remove your Samba drive as a shared resource in OSMC, it's pretty simple. On OSMC, go to Videos>Files and highlight – but don't click/enter – your Samba drive. Open the context menu (C on your keyboard) and select Remove source. You'll be asked if you're sure and if you want to remove all of the references to the media. If you never intend to use the Samba drive again, you can remove the references.

Set up a wireless AirPlay speaker

Combine a Raspberry Pi, a Wi-Fi dongle and your speakers with an amplifier to create a wireless stereo system

What you'll need

- Latest Raspbian image
  raspberrypi.org/downloads
- Wireless (or wired) network
- USB wireless dongle
- Speaker system
- USB sound card (optional)
- Small speaker amp (optional)

Every house needs a good speaker system, and a good speaker system is even better if it can be accessed by multiple devices over the wireless network. AirPlay uses Apple technology that was reverse-engineered in 2011, which means that third-party devices can now participate in the fun. AirPlay allows any Apple device to broadcast whatever is coming out of its speakers to an AirPlay receiver (which will be our Pi in this case). There is a way to send audio from PulseAudio to AirPlay receivers, as well as an application for Android called AirAudio. Sadly, the Android application requires root to access the audio of other applications running on the device. To keep things simple, we're going to use an iPad as the test client.

01 Set up Wi-Fi

Connect everything to your Raspberry Pi and power it up. Log in to the Raspbian system with the username pi and the password raspberry. Then, make sure your wireless interface is working by typing `iwconfig`. You should see an entry there for wlan0. If you look in /etc/network/interfaces, there should already be a section that looks like this:

```
allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

If it doesn't, then add it at the end. This configuration makes the wireless adapter connect once the network in wpa_supplicant.conf is in range. This means the only thing left to do is to edit the wpa_supplicant.conf file, which needs your SSID (wireless network name) and the passphrase. You can do this with:

```
sudo bash -c "wpa_passphrase my_network my_passphrase >> /etc/wpa_supplicant/wpa_supplicant.conf"
```

...which will result in a wpa_supplicant.conf file that looks like this:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
        ssid="my_network"
        #psk="my_passphrase"
        psk=17661426f1af334010ad2058d8b8f583ec501....
}
```

The easiest way to get things going is to reboot the Pi with **sudo reboot**. Once your Pi is back up, check that you have an IP address on wlan0 with **ip addr**.

02 Install Shairport dependencies

These instructions are based on a write-up by Drew Lustro (**http://drewlustro.com**). We need to compile Shairport, but before we can do that we have to install its dependencies:

```
sudo apt-get update; sudo apt-get install git libao-dev libssl-dev libcrypt-openssl-rsa-perl libio-socket-inet6-perl libwww-perl avahi-utils libmodule-build-perl libasound2-dev libpulse-dev
```

Now we have to compile a Perl module called Net::SDP.

```
cd perl-net-sdp/
git clone https://github.com/njh/perl-net-sdp.git perl-net-sdp
sudo bash -c "perl Build.PL; ./Build; ./Build test; ./Build install"
cd ..
```

03 Compile and install Shairport

Now we get to compile and install Shairport, which might take a minute or so:

```
git clone https://github.com/abrasive/shairport.git
cd shairport/
./configure; make; sudo make install
```

After this, the Shairport binary will be installed but it won't be started at boot, which is unhelpful. To fix this, we need to install the init script:

```
sudo cp scripts/debian/init.d/shairport /etc/init.d/shairport
sudo chmod 755 /etc/init.d/shairport
sudo update-rc.d shairport defaults
```

Finally, Shairport needs its own user that is a member of the audio group. You can create one with:

```
sudo useradd -g audio shairport
```

Above Here, the Pi speaker system and all of its components have been turned into one single, albeit messy, unit

04 Change the hostname

Shairport will show up as whatever the hostname of the system is. If you don't want the name to be raspberrypi, then you can change it like so:

```
sudo bash -c "echo PiPlay > /etc/hostname"
sudo hostname PiPlay
sudo sed -i 's/raspberrypi/PiPlay/g' /etc/hosts
```

05 Set the default sound card

If you have a USB sound card then you'll want to make that the default. You'll also want to make sure the volume is set to 100 per cent so that your amplifier is getting a nice signal. Use **aplay -L** to list the audio devices on your system. Our expert's sound card had an entry like this:

```
front:CARD=U0x41e0x30d3,DEV=0
    USB Device 0x41e:0x30d3, USB Audio
    Front speakers
```

To set that card as default, you do:

```
sudo bash -c 'echo pcm.!default front:U0x41e0x30d3 > /etc/asound.conf'
```

You can set the volume by going into alsamixer, pressing F6 to select the appropriate sound card and then using the up and down arrows to change the volume. Once you are done, you can make the changes the default with **sudo alsactl store 1**, where 1 is the card number. If you are using the built-in sound card then you should use 0 instead.

Finally, you can test the sound card is working with **speaker-test -c 2**, which sends a test signal to the left speaker and then the right speaker. You can exit the test with Ctrl+C.

06 A final reboot

After this reboot, Shairport should start automatically. To test it out, get your client and see if it can find the PiPlay. On an iOS device, slide up from the bottom, tap the AirPlay button and select PiPlay. It might take a couple of seconds to buffer, but once it's playing it should be fine.

07 Make it one unit

This is the freestyle part of the tutorial. Our expert decided to turn his setup into a single unit. You can see that in the image at the top of this page. Obviously, it could always be tidier, but that would involve cutting cables and we didn't have any spares.

Access point Chris turned his Ethernet-connected Pi into a wireless access point. He could then control the fireworks display using his iPhone

Power supply A 12V sealed lead acid battery goes into the switch box; the output of one switch powers the Pi, while the other goes through the relays

Shift register There are eight relays on each module and each shift register can split a byte of data into eight bits, feeding it out serially

Relay module These are great for controlling small appliances and high-current equipment. Chris only needed one of these for the fireworks

Components list

- Raspberry Pi B
- AC/DC 9-48V to 1.8-25V 3 switching power supply
- 5V 8-channel relay module
- 74HC595 shift register

Left The fireworks were connected to the relay module and then lit using Chris' homebrew e-matches

Below For ease when heading to the show, and when installing as a sprinkler controller, Chris mounted the components on one board

Fireworks controller

Chris Osborn shows how your Raspberry Pi can light up the Fourth of July

How did you first get started with this project?
So, I'm building a sprinkler controller and I realised that, before I installed it as a sprinkler controller, I could use it for fireworks. It's actually installed as a sprinkler controller now.

What hardware did you decide to use to build it?
I just got a bunch of 8-channel relay boards from eBay and some power supplies that would take up to 48 volts in, and do a DC-DC step-down (but it will actually take AC in as well, because that's what I need for the sprinkler controller), and it was like $4 for the power supply and I think $7 a piece for the relay boards. I connected three relay boards for the sprinklers, but I actually only used one relay board to control the fireworks. So yeah, there's pretty much nothing to this. The only other thing I had to use was a shift register, a 74-595, to control the relay board. I don't think there's any other chips on it, and I did have to use a resistor for a pull-up to keep the relay board disabled until the software starts up, but otherwise there's nothing to it.

Walk us through the circuit – what exactly is going on?
The shift register is connected to the GPIO on the Pi and I think that uses four pins – Enable, Clock, Data and Latch – and so it just shifts the data out serially, one bit at a time, and since the 595 control has eight bits and I have eight relays on a single board, one shift register for one board, I just shift out which relays I want to turn on and then latch it in, and they turn on. The only thing that was a little bit tricky, which for the fireworks was not an issue, is that the relay board is Active Low rather than Active High, and that seems to be pretty common with all of those relay boards that are like that. And the relay board that I used, I got one that uses 5-volt coils so I didn't have to feed it with any other voltages; so the five volts that feeds into the Raspberry Pi is also the same power supply that powers the relay board.

How exactly is the Raspberry Pi controlling the relay boards?
It's just some really simple software that I reused from a long, long time ago, when I was working with a much more complicated setup, and I just say which relay I want to turn on and then it just shifts out the data and turns it on. I put a time delay on it this time around so that after five seconds, it automatically turns the relay off.

Did you reuse the web interface from an old project as well, or was it custom-made for this?
Yeah, that's what I did; I wrote a Python script that runs... well, because of the way the GPIO works on the Pi, any script that wants to talk to the GPIO has to be root, so I just made that into a separate little program and I just tell it which relays to turn on. And then the web [interface] just calls that, and I had to do a sudo on that so I actually opened up a slight security hole just so Apache could do a sudo to call the script that enables the relays, but, you know, the Raspberry Pi was doing that for maybe ten minutes in my back yard, where somebody would have to be standing right next to it with their phone to hack into it. But yeah, the web interface was just an old, old, old C program that I had lying around, and it's kinda clunky, but it was from the previous setup that I had where I was using a laptop, access point and the gigantic relay board, and all that stuff was hooked up together and it was just such a mess that the last time it didn't go right, I decided to not do anything with that for several years. So I just reused that because it was way faster to get going with – all I had to do was change it so it called the Python script rather than the old, separate C program that it was using in order to control the GPIO.

Can you tell us what is was that changed your mind, then – why did you revive the fireworks project?
Because I've been thinking about it since the last time, thinking that if I could simplify everything so it was all just one thing that I needed to bring out instead of four or five different things that I have to hook up, and so like I said I was planning to do the sprinkler project and I was planning to do it after the 4th of July, and then I realised, 'Wait a minute... these are just relays. If I have this assembled in time I could probably use it for fireworks.' I went ahead and put the whole together, and everything was on one board except for the battery.

This meant that the battery was the only thing that was separate, and I used a 12-volt sealed lead acid battery and fed that into the power supply, so all I had to do was bring down one board, one battery, and then jumper-cable those together. So it was so much easier and I figured it was worth trying again, and I would say it was definitely much easier to set up. It didn't work perfectly – there were some problems where several of the fireworks didn't light, and I'm not exactly sure why because I actually saw the e-match glow, so I know that part lit and I'm not sure why the fuse didn't light. So I'm thinking that next time around I'll probably put some resistors on the relay switches so that when the relays are activated, the nichrome won't get maximum current but limited current, which means that instead of vaporising instantly it'll stay hot for longer.

Like it?
If you fancy putting one of these together yourself, you can download Chris' circuit diagram (**bit.ly/1gq89uC**) and also his script for controlling the relays (**bit.ly/1Jy1f1h**), then just order the components listed on the other page.

Further reading
Chris uses his own e-matches to ignite the fireworks, essentially made from nichrome wire and paper matches. He's written a guide to making them (**bit.ly/1fLjzrT**) and you can watch the whole setup in action on YouTube (**bit.ly/1ODJAVr**).

Forecasting the weather with your Raspberry Pi

With Python and a Raspberry Pi, you can keep an eye on the weather and be prepared for the next big storm

Why Python?

It's the official language of the Raspberry Pi. Read the docs at **python.org/doc**

Being someone who lives on the east coast of Canada, the weather is something that is consistently a topic of conversation. The common wisdom is that if you don't like the weather, just wait five minutes and it will change to something else. While people have used Raspberry Pis to create lots of applications like Twitter tickers, I thought that this issue we could look at how to write up a weather ticker in Python.

There are many different services available that provide weather data through a number of different APIs. Some can be accessed directly through a URL, while others are a bit more complicated. Luckily, for the more complicated options there are wrappers written to make the data collection easier. We will look at several services and see how to pull data from a number of them. From there, you can decide how best to display all of this information. The first service we will look at is the one provided by Weather Underground (**http://www.wunderground.com**).

> There are many different services that provide weather data through a number of APIs

This service uses weather information collected by individuals from around the world, using data from personal weather stations. As with most services, you will need to get an API key in order to pull down the weather data for your ticker. You can purchase a developer key for free, as long as you don't download data more than 500 times per day or 10 times per minute. This should be adequate for personal use. If you need more than this, you can purchase an API key that covers more usage. Interacting with Weather Underground involves sending a request as an HTTP URL and receiving either a JSON or XML file back.

In the sample code, we pull back the data as JSON. The first thing you will need is your location identifier. You can request a location identifier by either latitude and longitude or by geolocating your current IP address. You can even do a search by place name. Once you have this, you can go ahead and start to make data requests. You can import the urllib2 and json modules to make the request and parse the output. Let's say you wanted to get the current conditions at Cedar Rapids. You could do this with the following Python code:

```
f = urllib2.urlopen('http://api.wunderground.com/api/YOUR_KEY/geolookup/conditions/q/IA/Cedar_Rapids.json')
```

This will return a JSON file. You can now load this data and parse it with

```
json_str = f.read()
json_parsed = json.loads(json_str)
```

The json_parsed variable will now contain all of the available current conditions, such as temperature or precipitation. There are many other data features provided by the Weather Underground, including weather alerts, a three-day forecast, a ten-day forecast, hourly forecasts and tide forecasts. There is also historical information, in case you need historical data for some other project.

The next service we will look at is that provided by forecast.io. Forecast.io aggregates weather data from several different sources around the world. It parses through all of these data sources and provides current weather conditions and a forecast for your location through an API over HTTP. The returned data can be a bit messy, so there are wrappers for many different environments, including Python. This wrapper is available on GitHub from Ze'ev Gilovitz (**https://github.com/ZeevG/python-forecast.io**). While you can download and install from source, you should be able to install it using pip with:

```
pip install python-forecastio
```

As with Weather Underground, you will need to go to **https://forecast.io** and get an API key in order to make requests. And, as with Weather Underground, this API key is free. Once you import the module, you can call 'load_forecast()' to get the data. The object storing the returned results contains everything available and has several access functions. For example, you can get the hourly forecast with the object function 'hourly()'. These access functions have functions within them to access parts of the sub-data. For example, you can look at the hourly temperature with:

```
byhour = forecast.hourly()
for hourlyData in byhour.data:
    print hourlyData.temperature
```

In most instances, the information available through the wrapper functions should be good enough. But, you may

have need of more control over your request. In these cases, the forecast.io module has a function called 'manual()'. You can use this to send a particular URL data request to the forecast.io service to pull back the exact information you are interested in.

The last option we will look at is the python-weather-api module. This module provides access to weather service providers such as Yahoo! Weather, weather.com and NOAA. This module is actually available as a package within Raspbian. You can install it with the command:

```
sudo apt-get install python-pywapi
```

You can also install it with pip. Once you have it installed, you can import it and request data from any of the three data service providers. The three main functions are:

```
pywapi.get_weather_from_yahoo()
pywapi.get_weather_from_weather_com()
pywapi.get_weather_from_noaa()
```

These functions essentially get all of the available information from these different servers in a single call. You can then parse the results to pull out the information you are most interested in. The results actually come back as XML data and are then parsed in the return object. You can then pull out the relevant data by using keywords, such as 'forecasts'. You should review the module documentation to see what information is available from each of the data sources.

Once you have collected the weather data that you were looking for, you need to display this information. The simplest is to just print it out. This works well if you are using a console as the interface. There are also several LCD options available if you want to make a self-contained weather reporting service. The exact code used to handle the LCD will vary by manufacturer, but they all have very good documentation available. Now that you have weather reports being served to you, you will no longer have any excuses for being caught unawares by a storm or for showing up to work drenched by the rain.

Full code listing

```
# To talk to Weather Underground we need
# to import modules to handle URLs and
# JSON data
import urllib2
import json

# The next step is to open a URL and
# read the data
f = urllib2.urlopen('http://api.wunderground.com/api/YOUR_KEY/
geolookup/conditions/q/IA/Cedar_Rapids.json')
json_string = f.read()

# Now you can parse the JSON data
# read off the information you need
parsed_json = json.loads(json_string)
location = parsed_json['location']['city']
temp_f = parsed_json['current_observation']['temp_f']

# To talk to forecast.io you need to
# import the forecastio module
import forecastio

# You need your API key and location
apikey = "YOUR_KEY"
latitude = 36.4
longitude = 46.567

# The next step is to load the forecast data
forecast = forecastio.load_forecast(apikey, latitude, longitude)

# You can print out the available hourly data
by_hour = forecast.hourly()
for hourly_data in by_hourly.data:
   print hourly_data

# You can also get summaries
by_day = forecast.daily()
print by_day.summary

# To use the Python weather API you need to
# import the pywapi module
import pywapi

# Getting the weather from any of the
# available sources is a single call
# You will need to find and use the
# appropriate location ID

weather_com_result = pywapi.get_weather_from_weather_com('10001')
yahoo_result = pywapi.get_weather_from_yahoo('10001')
noaa_result = pywapi.get_weather_from_noaa('KJFK')

# The data is now in a key/value pair
# structure, ready to read off and used
print weather_com_result['current_conditions']['text']
print yahoo_result['condition']['text']
print noaa_result['weather']
```

Weather station

You can use Python to log weather data and send it in to Weather Underground; just install the required module with: `pip install weather`. This actually contains three separate submodules that need to be imported individually. The first provides a number of conversion functions, as well as calculation functions such as 'weather.units.calc_wind_chill()'. The second, 'weather.stations', provides the capabilities to talk to the weather station over a serial connection, but this module currently only talks to the Vantage and VantagePro. The third, 'weather.services', provides the functions to upload your data to the online services.

Send an SMS from your Raspberry Pi

Create a program that combines Twilio and simple Python code to enable you to send an SMS (text message) from your Pi to a mobile phone

Text messaging, or SMS (Short Message Service), has become a staple of everyday communication. What began life as a 40 pence message service is now offered by most tariff providers as an unlimited service. Twilio, a cloud communications company, enables you to send SMS messages for free from your Raspberry Pi to a mobile phone using just six lines of code.

01 Set up your Twilio account

The first step of this project is to register for a Twilio account and Twilio number. This is free and will enable you to send an SMS to a registered, verified phone. Once signed up, you will receive a verification code via SMS to the registered phone. When prompted, enter this onto the Twilio site to authenticate your account and phone. Go to **twilio.com/try-twilio** and create your account.

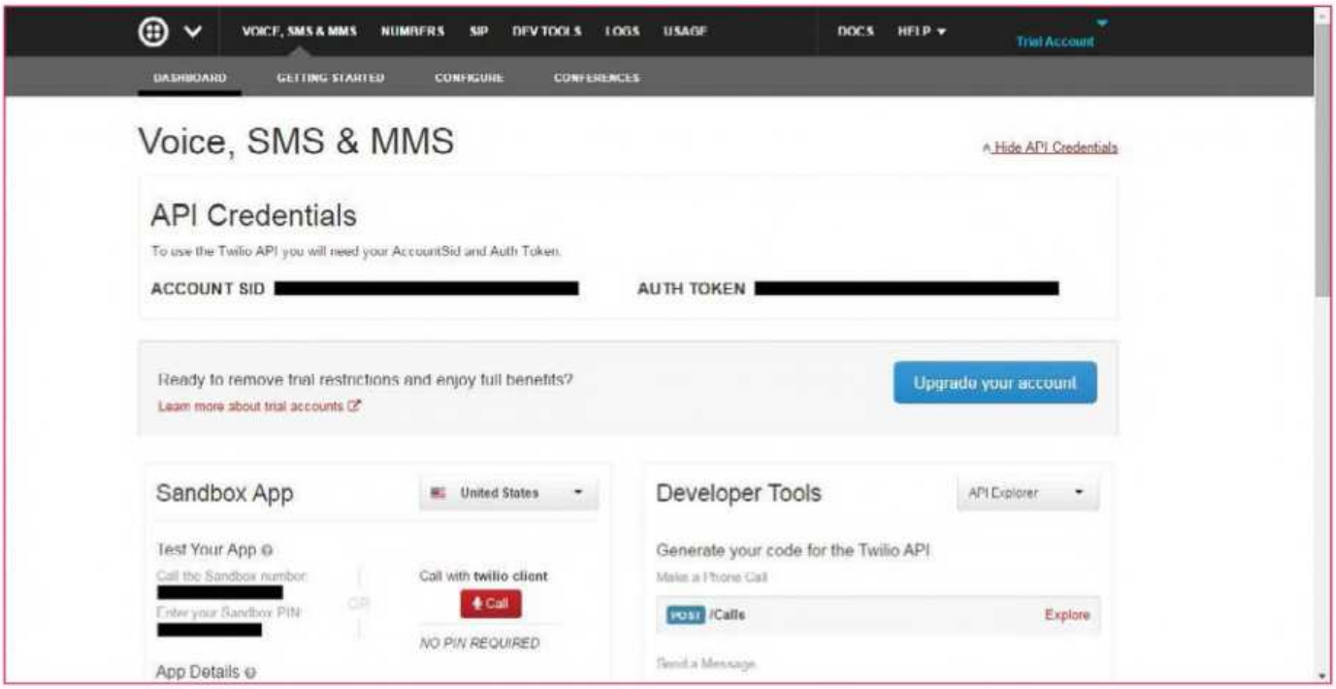
02 Register and verify mobile numbers

Your Twilio account is a trial account (unless you pay the upgrade fee), which means you can only send and receive communications from a validated phone number. Enter the phone number of the mobile that you want to verify, ensuring that you select the correct country code. Twilio will text you a verification code. Enter this code into the website form and press submit.

Left With this method, you could get your Pi to drop you a text when it finishes running a script

What you'll need

- Raspberry Pi
- Twilio account



03 The dashboard

Once registered and logged in, visit the dashboard page, which will display your AccountSid and your Auth Token. These are both required to use the Twilio REST. Keep these secure and private, but be sure to make a note of them as you will need them for your Python program later.

04 Install the software

Boot up your Raspberry Pi and connect it to the Internet. Before you install the Twilio software, it is worth updating and upgrading your Pi. In the LX Terminal, type **sudo apt-get update**, then **sudo apt-get upgrade**. Once complete, type **sudo easy_install twilio** or **sudo pip install twilio** to install the software. (If you need to install pip, type **sudo apt-get install python-pip python-dev**, press Enter, then type **sudo pip install -U pip**.)

05 Twilio authentication

Now you are ready to create the SMS program that will send the text message to your mobile phone. Open your Python editor and import the Twilio REST libraries (line one, below). Next, add your AccountSid and Auth Token, replacing the X with yours, as you will find on your dashboard:

```
from twilio.rest import TwilioRestClient
account_sid = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
              # Enter Yours
auth_token = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
              # Enter Yours
client = TwilioRestClient(account_sid, auth_token)
```

06 Create your message

You will probably want to be able to change your text messages rather than send the same one. Create a new variable in your program called **message**. This will prompt you to enter the phrase that you want to send to the mobile phone. When the program runs, this is the message that will be sent:

```
message = raw_input("Please enter your message")
```

> Twilio provides a wide range of API codes and reference documents to create other communication programs

07 Add your numbers

To send the message, you need to add the code line below and your two phone numbers. The first number is your mobile phone number, which is registered and validated with Twilio (Step 2). The second number is your Twilio account number, which can be retrieved from your dashboard page under 'Call the Sandbox number'. Change the Sandbox number to your country location and remember to add the international country code.

```
message = client.messages.
  create(to="+44YOURMOBNUMBER",
  from_="+44YOURTWILIONUMBER", body=message)
```

08 Send the message

Now send your message. The code below is not required, but useful to indicate your message has been sent. Add the lines and save your program. Ensure your Raspberry Pi is connected to the Internet and that your mobile is on, then run your program. You have just texted from your Raspberry Pi!

```
print message.sid
print "Your message is being sent"
print "Check your phone!"
```

09 Other API and codes

Twilio provides a wide range of API codes and reference documents to create other communication programs, such as making phone calls, recording a call, and retrieving data including caller IDs and call duration. The API also complements a wide range of languages, including Ruby, PHP, Java and Node.js (**twilio.com/api**). ■

Above You will be able to find your AccountSid and your Auth Token on the Twilio dashboard

REST

REST stands for Representational State Transfer. (It is sometimes spelt "ReST".) It relies on a stateless, client-server, cacheable communications protocol – and in virtually all cases, the HTTP protocol is used. REST is an architecture style for designing networked applications.

Working with RSS feeds

Learn how to build a feed ticker with your Raspberry Pi and a display to keep track of your social media feeds

Why Python?

It's the official language of the Raspberry Pi. Read the docs at **python.org/doc**

There is a vast amount of projects around the Internet that are centred around the Raspberry Pi as the engine for ticker-type displays. In this way, you can keep track of all of your Twitter or Facebook feeds. In these pages, we will take a look at another ticker service, specifically RSS feeds. While we could start with first principles, looking at making raw network connections and parsing the returned RSS data, that is a bit beyond the scope of such a short article. Instead, we will look at using the Python module 'feedparser', or the 'Universal Feed Parser'. This module will abstract out the lower level complications and enable us to focus on actually playing with the RSS data. True to its name, the Universal Feed Parser can work with most feed formats currently in use; this includes multiple versions of RSS, multiple versions of Atom and even CDF. Installation should be easy for most people. If you are using something like Raspbian, you can install it with:

```
sudo apt-get install python-feedparser
```

If you are using something else, you can always use pip and install it directly from Pypi.

Once you have feedparser installed, you need to start by defining the feed you want to read. While feedparser can read input from a file, or even a string object, the source we are most interested in is reading the RSS data from a URL. This way, we can get access to the most recent entries. The most basic form this takes looks like:

```
import feedparser

feed1 = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
```

This gives us an object that allows us to start accessing the data provided by this RSS feed. For example, you can get the title of this feed from the element **feed1['feed']['title']**. The feed element of the returned object contains information about the RSS channel as a whole. This includes the title, other items like the description, the feed link and the published date for the feed. This is good for the portions of your display where you indicate where a particular entry came from. The actual entries are available as a list, stored as the 'entries' element of the 'feed' object. Since this is a list, you can access the individual entries with all of the usual list syntax, such as slices. To save on typing, you may want to create a new reference to this list of entries, with something relating to:

```
entries_list = feed1['entries']
```

Each of these entry elements has a number of values available. The individual articles have a title, a link URL and a description along with a summary of the article. In most cases, these article entries will have embedded markup of one kind or another. There is also a published date, useful for if you wanted to filter out specific articles based on the date. Depending on how you want to have the display, you could just loop through and print the article titles. Or, you may want to print out the article summaries.

Now that we have collected the information, we need to display it. If you wanted to use an LCD display to have a scrolling ticker readout, there are several modules available. The one we will use in the sample is RPLCD. This particular module was inspired by the CharLCD module from Adafruit Industries, which it wrote to support the LCD units it builds to add to your Raspberry Pi. In order to use this, you need an LCD module plugged into the GPIO pins. You can then write a string to the LCD with something resembling:

```
from RPLCD import CharLCD

lcd = CharLCD()
lcd.write_string(u'Test String')
```

This is pretty simple to use. You can just loop through your feeds, pull the article titles and summaries, then write them out to the LCD device. If you do not want to use an LCD display, or would simply rather use a more traditional display, you can use a banner module to get the same rolling ticker type of display. The classic command line utility on Linux to do this type of display is **figlet**. In our examples here, we will use a port of figlet called **pyfiglet**. You can select one of the multiple fonts and print out the feed data to the screen. Then to use it within your Python program, you can use something like:

```
from pyfiglet import Figlet

f = Figlet(font='slant')

print f.renderText('sample text')
```

This means you can use almost anything for a display, as long as it has an HDMI connection. There is a massive number of fonts available to change and personalise the look of your very own RSS feeds. You may need to inform pyfiglet how wide the physical display is, in some number of characters. The default is 80 characters, which is the default on most terminal applications. If you are using a smaller screen, this is something you may need to think about how to deal with.

Now that we have looked at the Universal Feed Parser, you have no reason to miss any of the important news that may come your way across the Internet. And since we are playing around with text and font, you can feed this news to whatever kind of display you would like. This could be something like a full monitor screen or just a single line of notifications display. The really nice thing is that you can fit your original Raspberry Pi news feed display just about anywhere you like, or build it into a more portable setup. Hopefully, this will inspire you to add your own personalised news reader to your home or workspace.

Full code listing

```
import feedparser  # Reading RSS feeds

# The first thing to do is to read the RSS feed
feed1 = feedparser.parse('http://feedparser.org/docs/examples/atom10.xml')
feed1['feed']['title']  # The title of the feed is available
feed1['feed']['link']  # You can get the originating link for the feed
feed1['feed']['description']  # The feed has an overall description
feed1['feed']['published']  # The publication date is also available

articles = feed1['entries']  # The articles are available as a list
articles[0].title  # Each article has a title
articles[0].description  # and a description
articles[0].published  # and the publication date
articles[0].summary  # The summary is probably useful, too

# You can display the article titles on an LCD
from RPLCD import CharLCD
lcd = CharLCD()
lcd.write_string(articles[0].title)

# Or you can display on the monitor
from pyfiglet import Figlet
f = Figlet(font='slant')
print f.renderText(articles[0].title)

--------------------------------------------------

# Generating RSS feeds
import datetime
import PyRSS2Gen

rss = PyRSS2Gen.RSS2(
    title = "MY Raspberry Pi",
    link = "http://www.example.com/Python/PyRSS2Gen.html",
    description = "This is an example RSS feed",
    lastBuildDate = datetime.datetime.now(),
    items = [
       PyRSS2Gen.RSSItem(
         title = "Article 1",
         link = "http://www.example.com/news/030906-PyRSS2Gen.html",
         description = "This is the first article",
         guid = PyRSS2Gen.Guid("http://www.example.com/news/030906-
PyRSS2Gen.html"),
         pubDate = datetime.datetime(2003, 9, 6, 21, 31)),
       PyRSS2Gen.RSSItem(
         title = "Article 2",
         link = "http://www.example.com/writings/diary/
archive/2003/09/06/RSS.html",
         description = "This is the second article",
         guid = PyRSS2Gen.Guid("http://www.example.com/writings/diary/
archive/2003/09/06/RSS.html"),
         pubDate = datetime.datetime(2003, 9, 6, 21, 49)),
    ])
rss.write_xml(open("pyrss2gen.xml", "w"))
```

Sharing your RSS feed

There's a flip side to reading RSS feeds to keep up with the news: publishing RSS from your Raspberry Pi so that everyone else can keep up with what is going on with you. While you can craft all of the required meta information that wraps your feed and all of its articles by hand, this is not necessary. You can import the Python module PyRSS2Gen, which wraps all of the formatting work for you. There is a core function, named **PyRSS2Gen.RSS2()**, that creates your formatted RSS feed. It takes a number of named parameters that enable you to set all of the metadata. You can set the title, a link and a description, and the parameter **items** takes a list of the individual article entries. Each of the items also needs to be formatted, and this is done with the function **PyRSS2Gen.RSSItem()**. You can set the metadata for the item with a title, link, description and a publication date. Once everything is properly formatted, you need to dump this out so that it is available to other people. The object returned by the **PyRSS2Gen.RSS2()** has a **write_xml()** function to dump the final XML file for your feed. You need to give it a file handle to write to, so you could use something like:

```
rss.write_xml(open("pyrss2gen.xml", "w"))
```

... to dump the feed contents to. If you have regular information on your Raspberry Pi that you want to provide to the outside world, you can set up a cronjob to keep it updated. You will also need to have some way to make the feed visible to the outside world. You can do this by setting up a web server or at least putting the file up in an accessible location – this might be a file service, like Dropbox or an equivalent.

Hack your TV with Pi

Use the Energenie IR board and a Raspberry Pi as a remote control for your television

In this project you will learn how to emulate your television remote using your Raspberry Pi and a Energenie IR board in order to control your big screen. Why? So you can change the channel while no one is looking! Infrared, or IR for short, is light with a wavelength greater than the red end of the visible light spectrum, but less than that of microwaves. Infrared radiation can't be seen with the naked eye, but it can be felt as heat energy. Infrared radiation is used to transmit data from device to device, including between remote controls and their televisions, Blu-Ray players or to provide data links over short distances between computers or mobile phones.

This tutorial will show you how to set up the Energenie's Pi-Mote IR board, which will enable your Raspberry Pi to learn infrared remote-control signals and then transmit these same commands in order to control your device. This tutorial will focus on controlling a television in particular, but remember that the board is compatible with a wide range of devices. This means you can turn your Raspberry Pi into a remote for a range of household objects, such as your media system, smartphone dock or even the air conditioning.

01 Edit the config.txt file

Attach your IR board module to your Raspberry Pi and boot it up. It is always advisable to update your SD card software. In the LXTerminal type:

```
sudo apt-get update
sudo apt-get upgrade
```

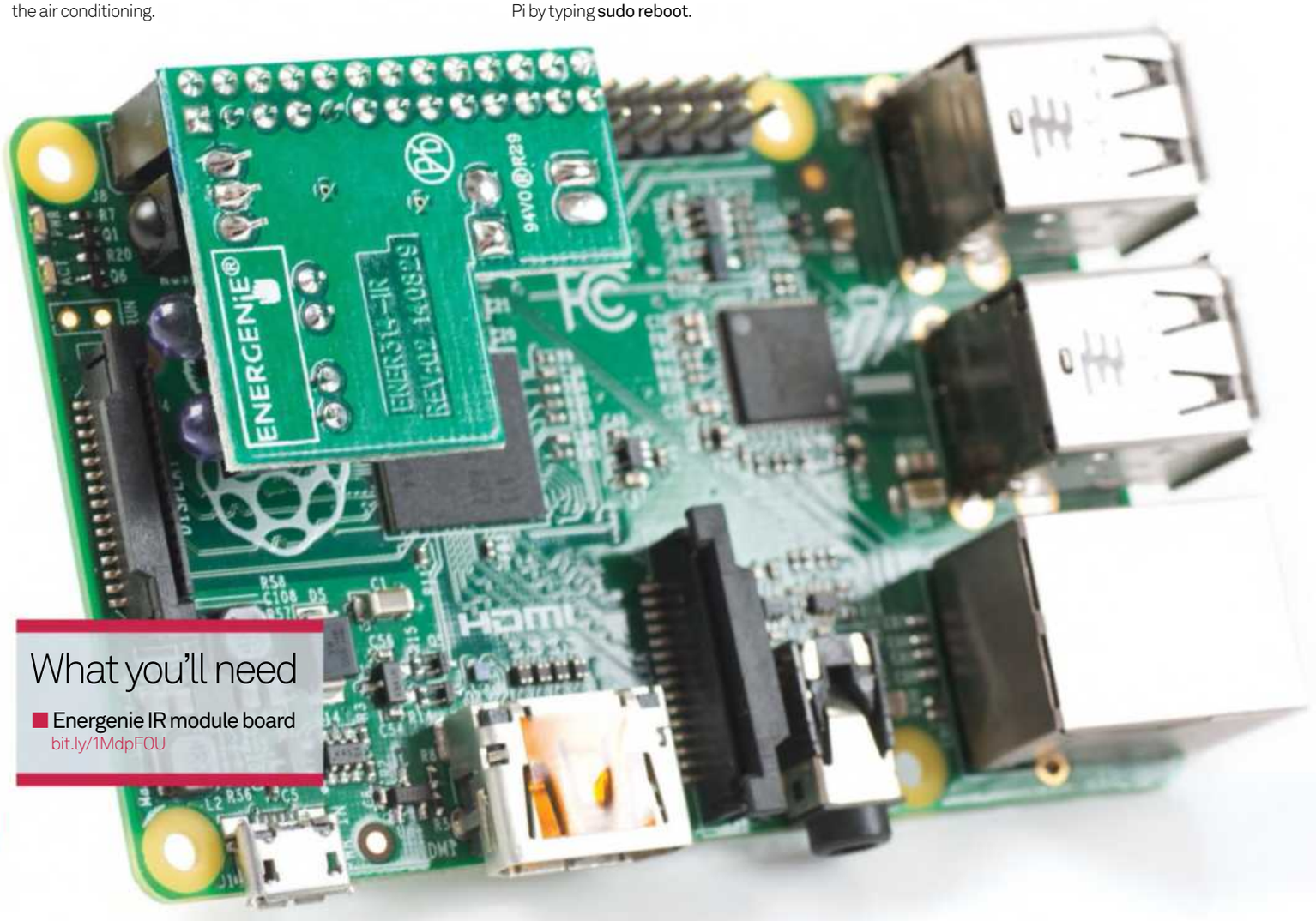
Next add code to the /boot/config.txt file to enable the LIRC IR software and IR module to interact. In the LXTerminal type:

```
sudo nano /boot/config.txt
```

This will load the config .txt file. Scroll to the bottom of the text and add the following line:

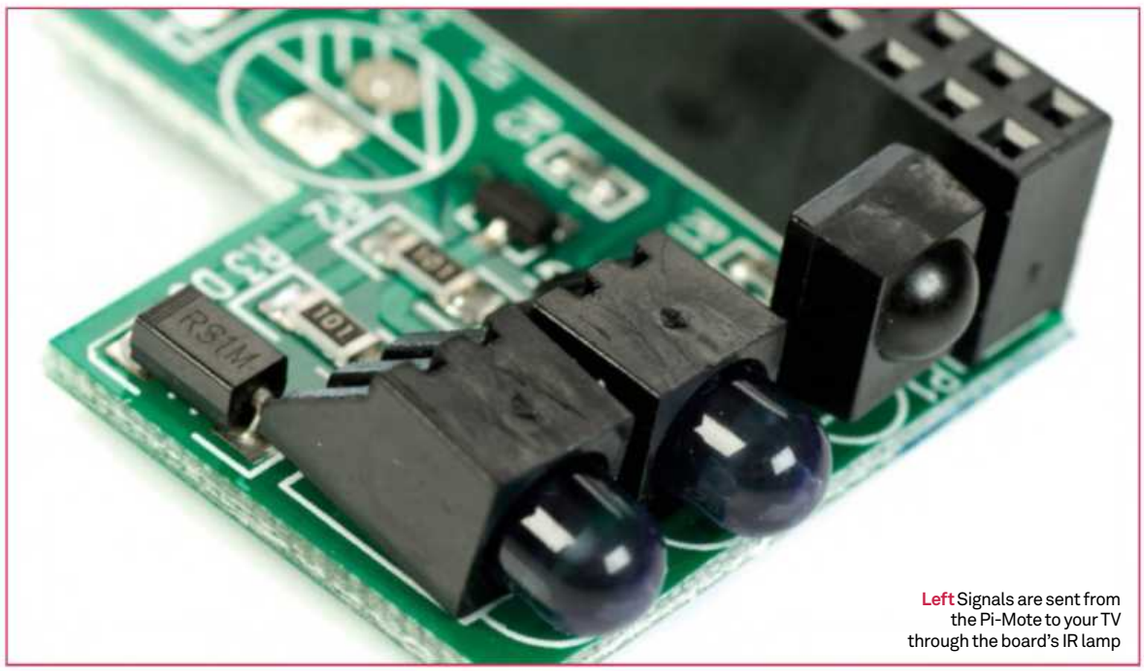
```
dtoverlay=lirc-rpi-overlay
```

Now press Ctrl+X and save the file, then reboot your Raspberry Pi by typing **sudo reboot**.



What you'll need

- Energenie IR module board
  bit.ly/1MdpFOU



Left Signals are sent from the Pi-Mote to your TV through the board's IR lamp

Lircd daemon

The LIRC enables you to decode and transmit infrared signals. The one used in this project is the lircd daemon that decodes IR signals received by the device drivers and accepts commands for IR signals to be sent if the hardware supports this. You could adapt your project to create a remote control for your media device using a simple Apple remote control as the input.

02 Install the software

Next install the LIRC software; this stands for Linux Infrared Remote Control. It is the program that enables you to interact with your television and transmit commands accordingly. In the LXTerminal type:

```
sudo apt-get install lirc
sudo apt-get install lirc-x
```

Restart your Pi. The default GPIO pins used when no pins are specified are pin 12/GPIO 18 for input for received infrared signals and pin 11/GPIO 17 for output for transmitted infrared signals.

03 What are all these files?

It's worth clarifying some file names and folders. All of the files used are stored in /etc/lirc, and there are two main ones: the hardware and LIRC configuration files (hardware.conf and lircd.conf). The latter holds all the data about your remote control, such as signal length, names of buttons and header details. This is the file to edit to emulate a remote control.

04 Test the IR receiver is working

To test that the IR receiver will 'pick up' the transmission from your remote, you need to stop the LIRC daemon, enable the test mode and then start the mode 2 testing. This runs a program to output the mark-space of the IR signal. It measures the pulse and space length of signals, returning the values to the terminal. Open LXTerminal and enter the commands below, then grab your control, point it at the IR receiver and press some buttons:

```
sudo /etc/init.d/lirc stop
sudo modprobe lirc_rpi
sudo mode2 -d /dev/lirc0
```

You should see something like this:

```
space 16300
pulse 95
space 28794
pulse 80
space 19395
space 28794
pulse 80
```

05 Transmit a signal

Now your remote is recognised and the IR board is working, you can use the **irsend** tool to record the signals and use these measurements to send commands to your TV. First alter the hardware.conf file located in the /etc/lirc folder:

```
sudo nano /etc/lirc/hardware.conf
```

Make the following changes:

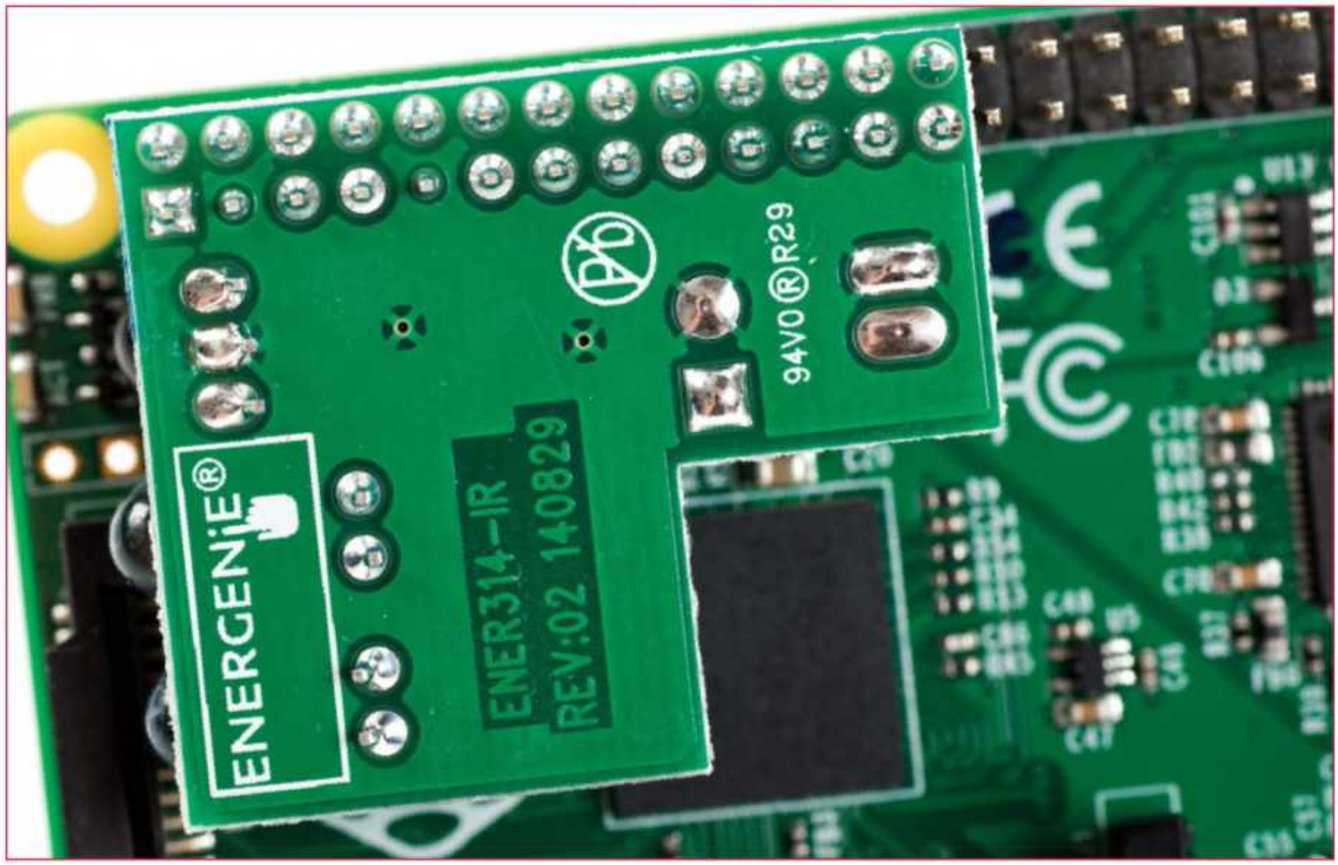
```
LIRCD_ARGS = "--uinput"
DRIVER = "default"
DEVICE = "/dev/lirc0"
MODULES = "lirc_rpi"
```

Press Ctrl+X to save the file – but don't rename it – then press Y and Return. Now restart the lirc daemon by typing:

```
sudo /etc/init.d/lirc restart
```

06 Get a new LIRC file

Next locate a compatible lircd.conf file that contains all the information you need to know about the remote's buttons and each of their specific functions. (Instead of buttons they are referred to as keys.) The good news here is that you do not need to create this file from scratch, because it is possible to use or adapt an existing remote control configuration file. These can be found on LIRC supported remote index websites, such as **lirc.sourceforge.net/remotes**. (Some remote configuration files will not be available on the LIRC index. This is because not all remotes are supported by LIRC. You may find more up-to-date files hosted on the GitHub website.) Download the correct file for your remote and save it in the Pi/Home folder. Note that we're using a Samsung remote in this tutorial.)



Above Energenie's Pi-Mote controller board costs £10, and you can get RC plug sockets with it for an extra £10

07 Use your new LIRC file

Once you have a suitable LIRC file, copy and paste over the code from the file that you have found into lircd.conf, which is stored in the /etc/lirc folder on your Pi. This will overwrite any current configuration setup that you have, but it is only an issue if you have already set up a configuration file. Open the existing lircd.conf file by typing:

```
sudo nano /etc/lirc/lircd.conf
```

Now copy and paste over the code from the new LIRC file. Press Ctrl+X to save the file, but do not change the name. Then restart the LIRC by typing **sudo /etc/init.d/lirc restart**. It will say that it failed to stop the daemon, but this is because it is already stopped and therefore it cannot be stopped again!

08 Test the lirc.conf file

Assuming that the lircd.conf file is compatible and your setup is successful, you can now test that it works. In the terminal window, type **irsend LIST Samsung " "** (replacing Samsung with the name of your television remote, which is stated at the top of the lircd file). This code will list all the KEYS (buttons) that are installed in the lircd.conf, displaying a list of the commands that you can send to your television.

09 Hack your television

Now that your lircd.conf configuration file is recognised, you are ready to control your television. The line of code is very simple and follows the format:

```
irsend SEND_ONCE Remote_Name Remote_Button
```

For example, to control the TV Menu you would type in the LXTerminal: **irsend SEND_ONCE Samsung KEY_MENU**. This will send the Menu IR signal and the menu will appear on the TV. In order to send a different button signal, alter the **KEY_** field. The key prefixes can be found in the lircd.conf file or by listing the keys with: **irsend LIST Samsung " "**.

10 Make your own lircd.conf file – part one

Sometimes you may not find a compatible lirc.conf file and instead you have to create one yourself. This involves running a program called **irrecord**, pointing your remote at the IR board and then simply pressing loads of buttons! This will then record the signals from your remote where you can assign KEYS to each of the signals. Stop the LIRC software by typing in the terminal:

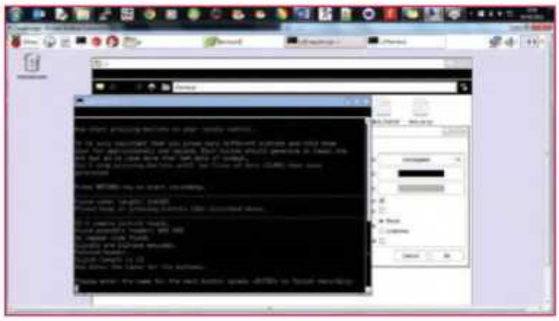
```
sudo /etc/init.d/lirc stop
```

11 Make your own lircd.conf file – part two

Next create a new lircd.conf configuration file and save the output. In the LXTerminal type:

```
irrecord -d /dev/lirc0 ~/lircd.conf
```

This will open the 'create' program and will present you with instructions on how to record and save the signals. There are two stages to this: the first part involves you repeatedly pressing the buttons on the remote until there are two lines of dots on the screen. This measures and records the signals being sent from the remote. Do this in a logical order, starting at the top of the remote and working downwards.



12 Make your own lircd.conf file – part three

Once the two lines of dots have been completed, your remote has now been recognised. The program will ask you to enter the names of the keys for each of the signals it has recorded. Follow each of the on-screen prompts, typing the names for each of the remote buttons/keys. For example, type KEY_UP and then press the corresponding 'up' key on the remote. You will then be prompted to type in the name of the next key, for example KEY_BACK, then press the 'back' key on the remote and so on. Keep doing this until you have entered names for each of the recorded keys.

13 Rename the remote

Unlike the ready-made lircd.conf files, the name of the remote on line 14 will probably be set as "home/pi/lircd.conf". Under the heading **begin remote**, find the **name** label and rename "home/pi/lircd.conf" as something else. This makes it easier to refer to in the code line, for example, as "Samsung".

14 Transfer the lircd.conf file

Now your lircd.conf file is ready to transfer to the /etc /lirc folder, as shown previously in Step 7. The simplest method is to copy and paste over the code that you have just created, but this will overwrite any old configuration file setup that you have. If you want to keep a previous configuration then follow Step 15, else jump to Step 16. In the LXTerminal, type:

```
sudo nano /etc/lirc/lircd.conf
```

Sometimes you may not find a compatible lirc.conf file and instead you have to create one yourself

15 Not overwriting

If you have already set up a lircd.conf file or you want to use a new one and still keep the old one, then you can create a new configuration file. This is automatically saved in the /home /pi folder and can be copied over to the /etc/lirc folder. Firstly, make a backup of the original lircd.conf file, creating a copy of the file and saving it as lircd_orginal.conf. In the LXTerminal, type:

```
sudo /etc/init.d/lirc start
sudo mv /etc/lirc/lircd.conf /etc/lirc/lircd_original.conf
```

Then copy over your new configuration file:

```
sudo cp ~/lircd.conf /etc/lirc/lircd.conf
```

Your original configuration file will be saved as lircd_original.conf.

16 Restart the LIRC

Now you have a configuration file you can use your remote control. Restart the LIRC by typing **sudo /etc/init.d/lirc restart**. As before, you can test that the lircd file is working by listing all the registered KEYS stored in the file; just type: **irsend LIST the_name_of_your_remote " "**. Send some commands to your television or device using the code:

```
irsend SEND_ONCE Remote_Name Remote_Button
```

For example, to control the TV Menu you would type in the LXTerminal: **irsend SEND_ONCE Samsung KEY_VOLUME_UP**. This will send the 'volume up' signal.

17 Common errors and code recap

Transmission error – this usually means that the lircd. conf file is not correct and contains an error. For example, using a file from 2007 instead of the 2015 version.

Connection refused – generally, this means the LIRC has failed or the hardware changes are not correct. Check the boot.config file and the hardware.conf file, then restart the LIRC by typing:

```
sudo/etc/init.d/lirc restart
```

Restart the LIRC program – this one can prove useful after you have changed a file, such as the hardware.conf or lirc.conf:

```
sudp/etc/init.d/lirc restart
```

Stop the LIRC program – useful when testing the program:

```
sudo /etc/init.d/lirc stop
```

Start the LIRC program –

```
sudo /etc/init.d/lirc start
```

List all the Keys in the file –

```
irsend LIST Samsung "" #
```

... replacing Samsung with the name of your remote.

Create a new bespoke lircd configuration file –

```
irrecord -d /dev/lirc0 ~/lircd.conf
```

Create a UI

It is possible to combine the IR board, the LIRC program and a web interface, which would open up many possibilities for projects. Combining the GPIO pins with a web server means that you can create a user interface that can be used to control your devices. Change the channel from your laptop or phone, turn the volume up or down or turn the TV off. A starter project can be found at **bit.ly/104CaMU.**

Make a visual novel with Python

Bridge the gap between books and videogames by creating an interactive novel or choose-your-own-adventure with Python and Pygame

Full code
FileSilo.co.uk

Videogames have come a significant way since their early days, and in recent years it's become even easier to tell a gripping and compelling story through the medium. A great way to tell a pure story is through the genre of visual novels. These interactive novels are extremely popular in Japan, though they're gaining traction in the rest of the world, and usually have the player click through a story and make decisions as they go to experience different plot points and endings.

In Python, this is a relatively easy project to create, but with the addition of the pygame module we can make it easier still and more expandable for the future. Pygame adds better support for positioning images and text, creating display windows, using mouse and keyboard inputs, and simplifying the coding process. We'll be coding this in Python 2, so make sure to run it in IDLE 2 and not IDLE 3 while writing and testing and coding.

What you'll need

- Python 2 python.org
- Pygame pygame.org/download.shtml
- IDLE Python IDE
- Game assets

Above Change scenes to add more depth to the story, and allow the game to have decisions and routes

Above Create an interactive menu to start a game or many games

These interactive novels are extremely popular in Japan, though they're gaining traction in the rest of the world

01 Get pygame dependencies

The best way to install pygame for your system is to compile it. To do this you need to first install the right dependencies. Open up the terminal and install the following packages, which in Ubuntu looks like:

```
$ sudo apt-get install mercurial python-dev
python-numpy libav-tools libsdl-image1.2-dev
libsdl-mixer1.2-dev libsdl-ttf2.0-dev libsmpeg-
dev libsdl1.2-dev libportmidi-dev libswscale-dev
libavformat-dev libavcodec-dev
```

02 Get the pygame code

Next we need to download the code for pygame direct from the source. Still in the terminal, you can do this by typing in:

```
$ hg clone  https://bitbucket.org/pygame/pygame
```

... which will download it to the folder pygame. Move to it using **cd pygame** in the terminal so we can continue building it.

03 Build the pygame module

To install it, we need to do it in two steps. First we need to prepare the code to install using the terminal with:

```
$ python setup.py build
```

One that's finished, you can then actually install it with:

```
$ sudo python setup.py install
```

This won't take too long.

04 Install in other ways

If the above steps don't work for you (or if it seems a little bit daunting) you can check the pygame website for binary and executable files that will work on other operating systems and Linux distros. Head to **http://pygame.org/download.shtml** to get the files you need for your specific system, including Windows and OS X. The rest of the tutorial will work in any OS.

Get more assets

You can add whatever game assets you wish – whether it's something you've drawn or other images you've managed to get online (that you have permission to use, of course). We like the website **opengameart.org**, which has a mixture of different assets you can use in various situations. You'll have to add more variables to the tuple that corresponds strings to variables and make sure you keep the names correct as you do it.

Right Use the mouse to move between lines and scenes to follow a story

Add music

Pygame does have a function for music, but depending on the libraries and codecs on your system you'll have different results with how it works. Either way, you can add music to punctuate scenes, make them happy, sad or foreboding. You can even add sound effects as well if you want to – we recommend expanding the elements of each line in the script file to add the information. You can also find royalty-free music and sound effects online if you don't have permission to use any others.

05 Get the visual novel files

We've uploaded the code to FileSilo, and here we're going to walk you through what we've done to make it work. Download the files for the visual novel and unzip them. The two files we care about for the moment are the visualnovel.py and script.py python files – this is where all the important code is.

06 Understand the script file

For the moment the script file is small and literally just holds the script for the game. It's made up of events for the visual novel to move between, line by line, by splitting it up into scenes. This includes the location of each line, the character, the actual line itself and information on how the game flows. These are all matrices that store the relevant information, and are completely customisable.

07 How the script relates

In our game, the code pulls in elements from the script file as it goes. We'll explain how that works later, but this also allows us to implement decisions later on to change where the game might take you in a later part of the game.

```
import pygame, time, script
pygame.init()
```

08 Starting the main game

We don't need many modules for the current state of our visual novel. Here we've imported the new pygame module, our script as a module and the time module for aesthetic reasons – we're going to have the code pause in bits rather than just instantly change scenes to the next line. We also initialise pygame with a simple **pygame.init()**.

09 Add variables and assets

We can now add the mixture of info that we need to run the novel. We define the size of the display screen to use (1000 pixels wide and 563 high), along with some RGB colours for the code to use. We're also telling pygame which font to use, and how large for certain sections, and also loading images for the game.

10 Start the game

Create a display for the game. Pygame works by constantly updating the display with new information. To show how this works, the **menu** function adds elements to the display (which we've titled screen), such as filling it with colour, adding shapes and using **blit** to add images, or in this case text. Once you've created a buffer of changes to the screen, you update it with the **flip()** function.

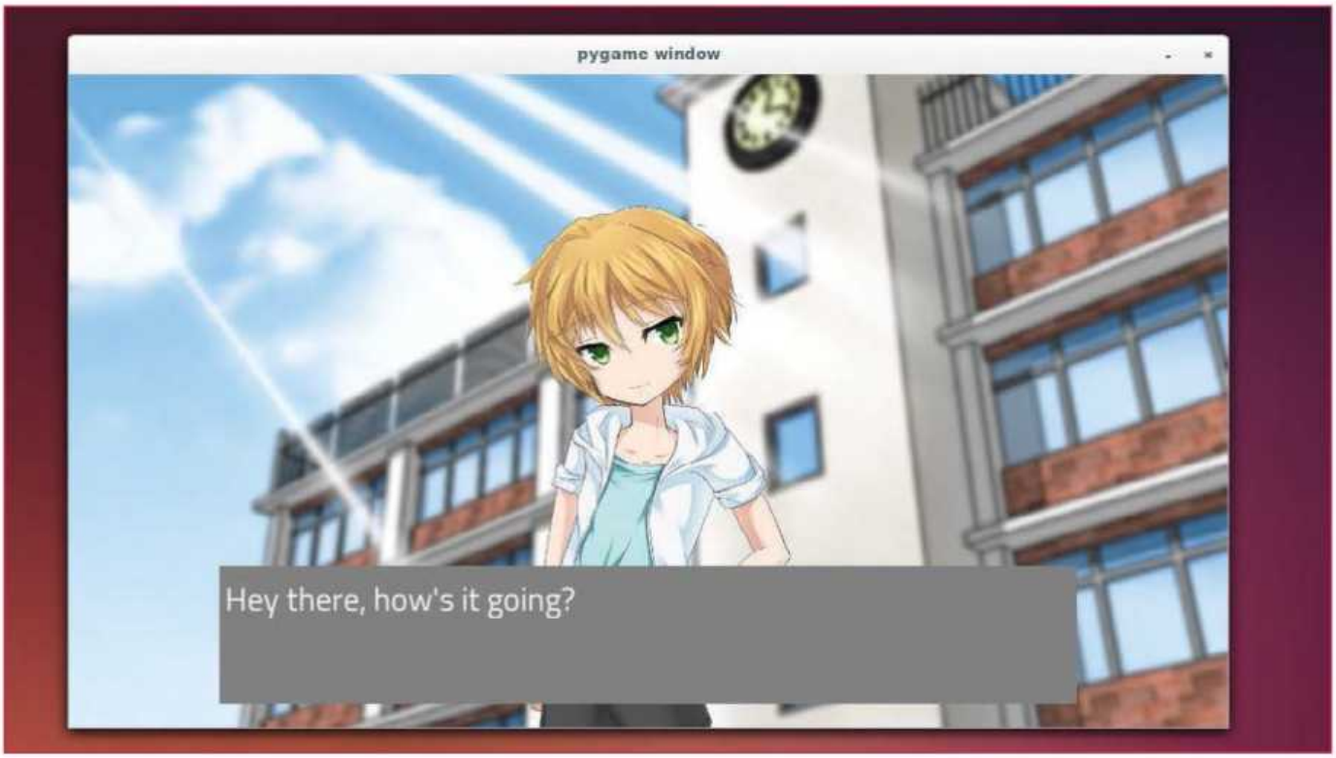
11 See the mouse

As we've created the button as a rectangle and now an image on the menu, we need to know that the mouse is hovering over it to know when it's clicked the button. First we have to use **event.get()** to see the mouse in general, then we look for the position with **get_pos()**. After that, we wait for it to click, see where it clicked (using the co-ordinates of the rectangle) and then make a decision after that.

```
def start_game():
    global game_script, location, character
    turn = 0
    game_state = 1
```

12 Start the story

Our **start_game** function is called when the mouse clicks the right position and we prepare the game, getting the characters, locations and progression through the game script. The rest of this function uses this info to pull in data from the script to make the game flow properly.



Above Creating believable dialogue for your game should be one of your primary creative concerns

```
line_start = 0
for i in range (4):
    if line_start == 0 and line[0] != '0':
        print line[0]
        screen.blit(location[line[0]], [0, 0])
        time.sleep(1)
    elif line_start == 1 and line[1] != '0':
        screen.blit(character[line[1]], [377, 113])
        time.sleep(1)
    elif line_start == 2:
        pygame.draw.rect(screen, (grey), pygame.Rect(130, 423,
    elif line_start == 3:
        screen.blit(game_font.render(line[2], 1, white), (135,
        turn += 1
        if  line[3] != '0':
            game_script = line[3]
            time.sleep(0.5)
        game_state = line[4]
        clicked = 0

    line_start += 1
    pygame.display.flip()
```

13 First screen

The first screen is handled differently to the rest, and acts to get every element up on the interface before we start continuing – it makes the code take up a little less time to process as we begin. The **getattr** allows us to use the string/integer associated with our place in the story and call upon the relevant scene function from the script file. We then use an if statement with an iterative function to successively add screen elements, to give the illusion that it's building up the first screen. We finish it by advancing the progression value.

14 Go to the next line

Similar to how our original startup code works, our next if statement and iteration checks to see what is different on the next line, and if it moves to a different scene function, and changes anything that is different without filling up the buffer any more than is needed. We've made it so no change is labelled with a 0 in the scripts.

> The code we've written is very expandable, allowing you to add decisions that are logged to take you to different scenes – or 'routes', in visual novel terminology

15 The starting function

We finish our code bit with a simple function that starts off the entire game. This is just to encapsulate the entire code and allows us to add different ways of turning it off in the future. When running the file, IDLE will load everything up and then run the **game()** function at the end – this is similar to how you can add a **__main__** function at the end that will start the code in the command line.

16 Expand your code

The code we've written is very expandable, enabling you to add decisions that are logged to take you to different scenes – or 'routes', in visual novel terminology – and make your game feel more interactive. This doesn't require much more code to the if statements, and is also a good way to look into adding graphical buttons to click in order to use the collide function.

17 Move the assets

Currently, the code has the script-specific assets in the main visualnovel file. These can be moved to the script, allowing you to make the visualnovel file much more modular so that can you have multiple scripts with different assets to load at startup. ■

Compile the game

Finished your game and want to send it as an actual game for people to play? Py2exe is a program that lets you do this, and you can grab it from here: **py2exe.org**. This enables people to play the game without needing to install Python or modules to their system.

Turn your Raspberry Pi into a stop-motion studio

Build your own animation studio by using your Raspberry Pi as a stop-motion camera

What have you done with your Raspberry Pi camera lately? While it gives us plenty of new ways to use the Pi, unless you've got your computer set up as a security webcam or you're particularly a fan of time-lapse photography, the chances are that you've overlooked the Pi camera module for a while.

If you're a fan of animation or you simply want to extend the possibilities of the module, why not build a stop-motion camera? By using Python and an external button to capture images, the Raspberry Pi can be the perfect tool for animators.

Better still, you can go beyond animating toys or bits of LEGO and go old school by mounting the Pi on a rostrum and creating a cartoon. Even if you can't buy or build one, you can mount the stop motion Pi camera with a smartphone mount for stability.

01 Mount your stop-motion Pi camera

Before you get started, think about the type of animation you're going to be capturing. If you're using the traditional top-down method, as used by classic cartoon animators, then you'll need a rostrum to mount the Raspberry Pi.

Alternatively, you may be animating something on a desk, table or perhaps the floor, but you'll need your Pi camera mounted in a similar way, looking across rather than down.

Various options are available, such as smartphone tripods and dashboard mounts. Most of these should be suitable for securely mounting your Raspberry Pi.

02 Find somewhere to shoot

For your first attempts at shooting a stop-motion video, you should use a wide and uncluttered space. This might be a desk, a kitchen work surface or even the floor, but it should be a hard and flat area in most cases (unless you have need for a bumpy carpeted environment for your video) to aid with the creation of your stop-motion film.

As time progresses and your skill develops, other surfaces can prove useful alternatives, but keep it simple for now and stick with flat surfaces while you get to grips with the art form using the Raspberry Pi stop-motion camera.

03 Connect the Pi camera module

Next you'll need to connect the Pi camera module to your Raspberry Pi. All models have the necessary connector, although where it is found on the device will depend on the version of your Raspberry Pi.

The Model A has the Pi-camera connector next to the Ethernet port, as does the Model B. On the B+ and the Raspberry Pi 2, the connector is in a similar position, but it's a little further from the Ethernet port between the audio-out and HDMI ports.

Connecting the camera module can be tricky. Begin with taking your Pi out of its case or remove the top where possible and disconnect all cables. Take precautions before removing the device from its antistatic bag, as the camera module is very sensitive to static electricity.

On the Pi, lift the plastic catch on the connector and slot the camera module flex into place with the shiny contacts facing away from the Ethernet port. Once the flex is fully slotted in, push the plastic catch back into place.

04 Test your Pi camera module

After connecting the Pi camera, check that it works by booting the Raspberry Pi (we're assuming you're running Raspbian) and entering this in the command line:

```
sudo raspi-config
```

With the keyboard arrows, move down to option five, 'Enable Camera', and tap Enter. In the following screen, hit Enter again to enable the camera and exit.

If you're not already signed into the GUI, do so now (if you're in the command line interface, enter **startx** to launch the desktop view). Open the terminal and enter:

```
raspistill -o image1.jpg
```

You can review the resulting image in your Home directory.

05 Straighten out the image

With the Pi camera up and running, you may notice that it's outputting the image with the axes flipped. We can fix this using Python, so open the terminal and enter:

```
sudo apt-get install python-picamera python3-picamera
sudo idle3
```

In the Python editor, open File>New Window and enter the code below, setting the camera.vflip and camera.hflip as True or False as required. Save (perhaps as 'camflip.py'), then press F5 to run the script and view the correctly outputted image.

To save time, however, you might try rotating the position of your camera or Pi camera module!

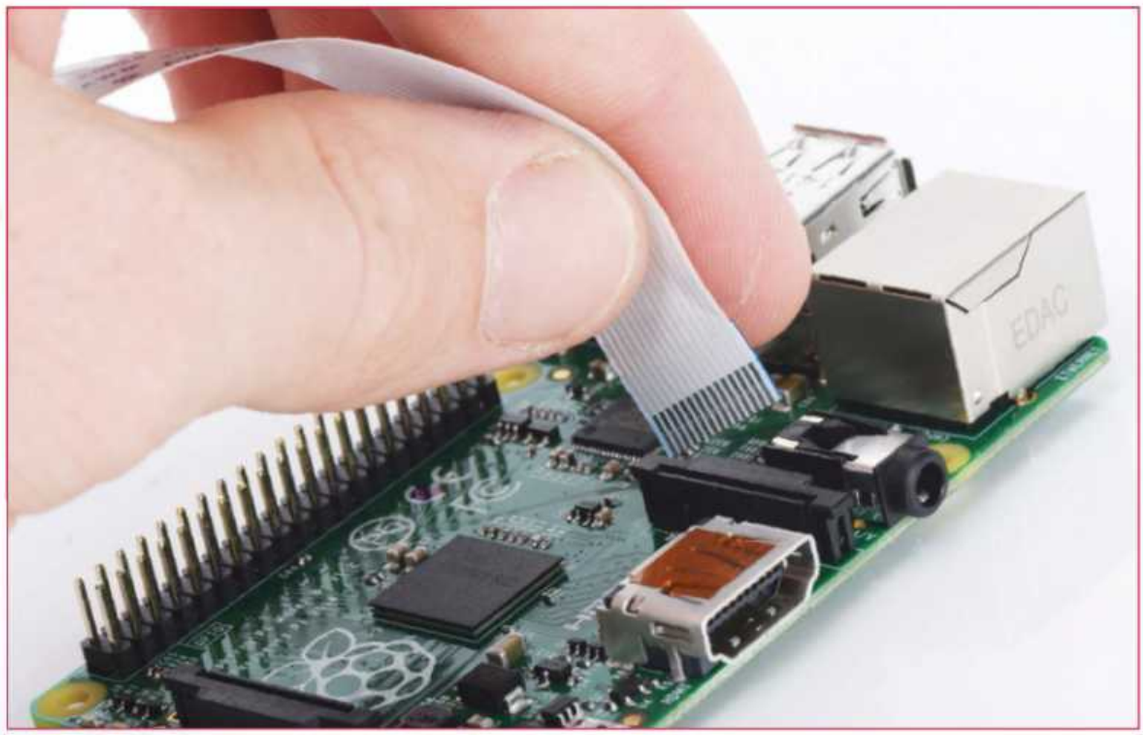
```
import picamera
from time import sleep

with picamera.PiCamera() as camera:
    camera.vflip = True
    camera.hflip = True
    camera.start_preview()
    sleep(3)
    camera.capture('/home/pi/image2.jpg')
    camera.stop_preview()
```

06 Set up the breadboard and button

We have two ways to add a button to the Raspberry Pi, but before proceeding, ensure you have switched the computer off and disconnected it from the mains. You should also disconnect any cables and hardware.

The simplest method of adding a button is to employ a solder-free breadboard and a single-state pushbutton. Connect the button to the breadboard with two male-to-female wires running to GPIO pins GND and 17. With a script designed to detect action from the button on the GPIO, each frame of your animation can be captured with a single button push.

> You may notice that it's outputting the image with the axes flipped. We can fix this using Python



Left Consider the angle you'll be shooting from as you are setting up

Right With the camera module, ensure the shiny side faces away from the Ethernet port

Tripods and suction holders

Don't want to build your own rostrum? Why bother when a camera tripod can be positioned as needed and other items, like smartphone suction holders and grips, can be employed to hold your Raspberry Pi case and camera module in place?

For top-down animation, suction-pad smartphone holders (available for under £10) that use a sticky gel for a stronger grip are perfect for holding your stop-motion Pi camera and attaching to a flat surface above the animation subject.

Don't want to build your own rostrum? Why bother when a camera tripod can be positioned as needed?

07 Code for stop motion

Once satisfied with the results of your Pi camera, it's time to turn it into a stop-motion camera. The first step is to type up the code shown below, which will capture an image of your subject and save it into a folder called 'Stop motion'. Each image is numbered sequentially and they can all be stitched together once your animation is complete. Save the code as animation.py:

```
import picamera
from RPi import GPIO

button = 17

GPIO.setmode(GPIO.BCM)
GPIO.setup(button, GPIO.IN, GPIO.PUD_UP)

with picamera.PiCamera() as camera:
    camera.start_preview()
    frame = 1
    while True:
        GPIO.wait_for_edge(button, GPIO.FALLING)
        camera.capture('/home/pi/animation/frame%03d.
                        jpg' % frame)
        frame += 1
    camera.stop_preview()
```

Then, in a new terminal window, enter the following:

```
sudo python3 animation.py
```

Press the button to capture each frame, moving the subject as needed. When you're all done, hit Ctrl+C to terminate the script.

08 Stitch together your stop-motion animation

The collected images can be cycled through relatively quickly using a special picture viewing app, but for a true animation you will need to compile them into one single file. In the terminal, install ffmpeg:

```
sudo apt-get install ffmpeg
```

After installing, you can then convert your images into a video clip, as follows:

```
ffmpeg -y -f image2 -i /home/pi/Desktop/stop-
motion/frame%03d.jpg -r 24 -vcodec libx264 -profile
high -preset slow /home/pi/Desktop/stop-motion/
animation.mp4
```

With this file created, open with the command:

```
omxplayer animation.mp4
```

The video will then be played in full-screen mode.

09 Use an app instead

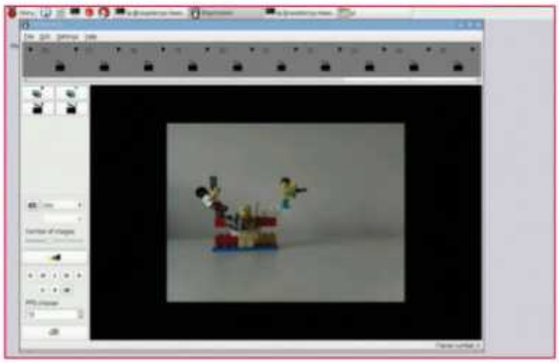
Don't fancy using the script? Try this stop-motion application. Begin by installing the raspicam-extras package that includes the UB4L drives for the Pi:

```
wget http://www.linux-projects.org/listing/uv4l_
repo/lrkey.asc && sudo apt-key add ./lrkey.asc
sudo sh -c 'echo "deb http://www.linux-projects.
org/listing/uv4l_repo/raspbian/ wheezy main" >> /
etc/apt/sources.list'
sudo apt-get update
sudo apt-get install uv4l uv4l-raspicam uv4l-
raspicam-extras
```

With that done, enter:

```
sudo apt-get install stopmotion
```

Launch with the **stopmotion** command to open a GUI with a live camera for you to line up each shot. This is a more elegant solution and captured images can be stitched together using the 'Number of images' slider and the camera button above it.



Above Here's the stopmotion program in action – it's a simple enough GUI to get your head around and gives you a nice preview window

10 Put it all together

Now you have the camera set up, a device for keeping it steady (whether a DIY rostrum or a tripod), and you've constructed a button or plan to capture each frame via SSH. Your stop-motion Raspberry Pi camera is finally ready!

By now you're probably aching to get started, so with your stop-motion Pi camera ready to use (and close to a power supply), it's time to start building your film set. While this might simply be an empty table top, there might equally be a few props you would like to include.

11 Storyboard your shoot

It's easy to get tied up with the idea of creating a stop-motion camera and forget all about a subject and how it will act.

You can avoid any problems here by taking the time to carefully plan what will happen in your film: your story. Remember, each second of the video will require 26 frames!

The best way to plan at this level is to simply write up an outline, but beyond this you may prefer to storyboard instead by making pencil sketches to help you progress the story.

12 Cast your stop-motion shoot

You'll also need a good idea of what your subject will be; this means who or what you're going to be using the stop-motion camera to capture frames of. Typically, amateur stop-motion films make use of household objects, toys and child's play clay.

The beauty of this kind of animation is that you can use almost anything that you can get your hands on, from a cup and saucer to an Action Man, as long as you have a way to support the subject(s) in the positions you wish them to take throughout.

13 Stop-motion with toys

If you cast toys as your stop-motion stars, you will get a much better result from something that is built to stand up on its own than toys that tend to sit or fall over.

LEGO sets and Minifigs appear in many stop-motion productions on YouTube. This is with good reason, as they're really easy to place in a desired position. The construction element of the bricks is also a major attraction. Another popular option is Transformers toys. These are both good places to start, but you should aim to develop your own approach over time.

14 People in stop-motion films

It isn't only inanimate objects that you can include in stop-motion films. People can feature too! Pop videos such as Peter Gabriel's 1985 hit *Sledgehammer* have taken advantage of stop motion (that video was produced by Aardman Animations, the eventual creators of *Wallace and Gromit*) and the technique can be used on humans to create surreal effects. If you want your subject to be moving around a room too, they can appear to be floating or gliding. The results can be strange, but useful if you know what you want.

15 Make your own *Wallace and Gromit*

Known as 'claymation', the practice of animating lumps of clay has been a popular form of animation for years in the UK, but there's more to it than just clay. These forms, whether they're cheese-loving old men or remarkably clever dogs, have a wire skeleton that is used to keep movement in the desired position.

This makes it much easier to capture the frames efficiently, but for the best results you should also have several versions of the same figures available. This is just in case one gets deformed and damaged during production!

16 From stop motion to time lapse

Similar to stop motion, time lapse is a technique that automatically captures images on a preset timer. We can use a Python script to control this, saving the captures in a directory and using ffmpeg to compile them into a film.

However, what you may not want for this project is a mains cable trailing all over, especially if you're attempting to capture the movement of the stars at night or nature activity. We suggest employing a Pi-compatible battery pack to make your time-lapse Pi camera truly mobile, using SSH to run the script remotely:

```
import time
import picamera

VIDEO_DAYS = 1
FRAMES_PER_HOUR = 60
FRAMES = FRAMES_PER_HOUR * 24 * VIDEO_DAYS

def capture_frame(frame):
  with picamera.PiCamera() as cam:
    time.sleep(2)
    cam.capture('/home/pi/Desktop/frame%03d.jpg' % frame)

# Capture the images
for frame in range(FRAMES):

  # Note the time before the capture
  start = time.time()
  capture_frame(frame)

  # Wait for the next capture. Note that we take into
  # account the length of time it took to capture the
  # image when calculating the delay
  time.sleep(
    int(60 * 60 / FRAMES_PER_HOUR) - (time.time() - start)
)
```

17 Take your stop-motion studio to the next level

At the risk of encouraging you to become the next Ivor Wood (creator of *The Wombles*, *Paddington* and *Postman Pat*, among others), it is possible to use the Raspberry Pi's camera module for ambitious projects as well as small ones. After all, this device photographs in high resolution so there is no reason not to adopt this setup and incorporate it into a working stop-motion studio with a miniature set.

Sharing your work through YouTube is a great idea too, especially as it will make it simple to add a soundtrack using YouTube's browser-based editor. ■

Access Twitter with Python

Twitter can be a great communication channel to enable your Raspberry Pi project to talk to the world

Earlier on in this bookazine, we looked at how you can turn your Raspberry Pi into an RSS feed reader and how to enable your Raspberry Pi to post an RSS feed for others to read. Now that you've mastered that, we will continue along the same lines and look at how to add the ability to have your Raspberry Pi post tweets for the rest of the world to follow what is happening.

The first step is to find some way to talk to Twitter. Twitter provides a full API that defines how you can send and read tweets. We could work with raw sockets; manually format and send messages, then process the returned results. However, this completely defeats the whole point of Python and the premise of code reuse. Instead, we will use the module python-twitter. There are several other options available too, in case you wish to look for a module that uses a different code style that better matches your own. They should all have the same functionality, so feel free to explore and experiment. For these examples, you can install the module with **pip install python-twitter**. This works fine for Python 2.x, but Python 3 support is something that is in progress, so you may need to pick a different module if you want to use Python 3. If you need the latest features then you can always grab the source code from GitHub and build it.

Once you have the module installed, you can import it within your code with **import twitter**. The class twitter.API provides the main interface to enable you to interact with the Twitter API. When you create a new instance of the class, you need to provide your credentials in order to authenticate to Twitter and be allowed to work with your account. Twitter now uses OAuth as the main way to handle authentication, and this means that you need to generate an access token that your code can use to identify itself with. Twitter has an entire section within its developer documentation that covers the steps required to get your token, located at **goo.gl/99qAfa**. The first step is that you need to register your code as an application that is authorised to talk with Twitter. You handle this by going to **apps.twitter.com** and creating a new app. This gives you the consumer key and consumer secret that identifies your code as being permitted to connect to Twitter. The second step is to create an authentication token for your account that will tell Twitter that this app is authorised to connect to your account, read it and post to it. You should end up with two separate keys and secrets that you can now use to create an instance of the API class. The code looks like:

```
import twitter
  api1 = twitter.API(consumer_
         key='consumer_key',
         consumer_secret='consumer_
         secret', access_token_
         key='access_token',
         access_token_secret=
         'access_token_secret')
```

... where the relevant keys and secrets are the ones that you have generated above. You can check to see whether these credentials worked by calling the function **api1.VerifyCredentials()**. This will either give you an error or give you a breakdown of the account information.

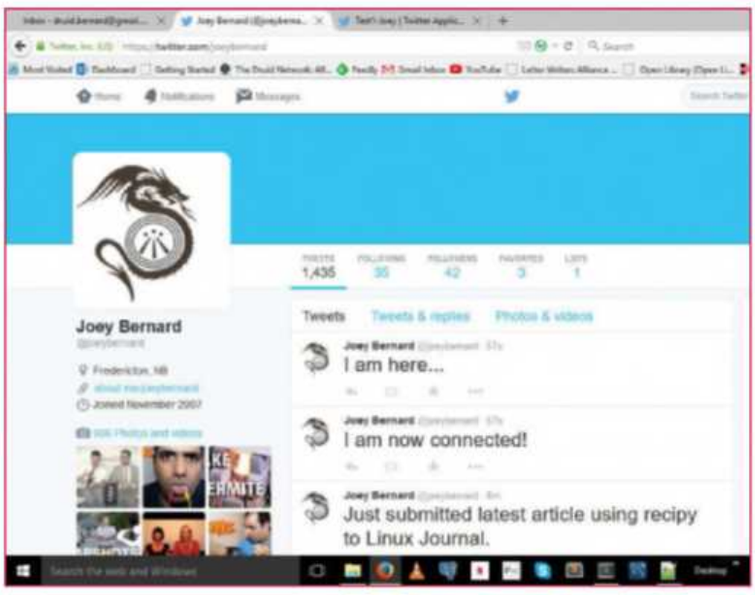
Once you have an authenticated connection to Twitter, your Raspberry Pi project can start to talk to the world at large. The most basic method is **PostUpdate()**. With this function, you can send in a status update message, with a maximum of 140 characters. If your message is longer than that, you can use the function **PostUpdates()**, which will post multiple status updates until the entire message is posted. If you want to, you can include other details with your message. This includes options like the current longitude and latitude, or a place ID, to identify where a particular update came from. This might be handy on a mobile monitoring project of some kind where you want to know where a particular update came from. You can also post an update as a response to some other message, by including the **in_reply_to_status_id=** option in the function call. If you don't want to use public status updates for your messages, you can use the function **PostDirectMessage()** to send a text message to a particular account. You can use either the **user_id** or the **screen_name** to identify the recipient of your direct message. While this is a bit more private than a public update, it is by no means secure, so don't send any compromising information using this method. If you want to, you can also send multimedia files as Twitter updates. The function **PostMedia()** can take either a local file name or an HTTP URL to do a multimedia tweet. You can send image files like JPEG, GIF or PNG. You can also include an associated text message for the tweet, as well as location information if you want to geolocate the tweet and image. Once you finish any of these posting functions, you get a status object returned that gives you details of your status. There are two dozen elements giving you items, such as whether it has been favourited, when it was created and the hashtags associated.

The Twitter API also enables you to read updates and direct messages from other users. This means that you can use this same communication channel to be able to send commands to your Raspberry Pi project. The function **GetDirectMessages()** can get up to the last 200 direct messages sent to your Raspberry Pi's Twitter account. Luckily, you don't have to do this every time. You can use the option **since_id=X** to tell Twitter to only send those direct messages that are newer than message ID X. If you are only interested in getting the last Y messages, you can instead use the option **count=Y**. This function returns a sequence of DirectMessage object instances. This structure contains the message ID, the creation date, recipient and sender information, as well as the actual message's text. This way, you can send commands within the text message so that your project can do different tasks after it has been deployed.

With the examples we have here, you should be able to have your project tweet what is going on out to the world.

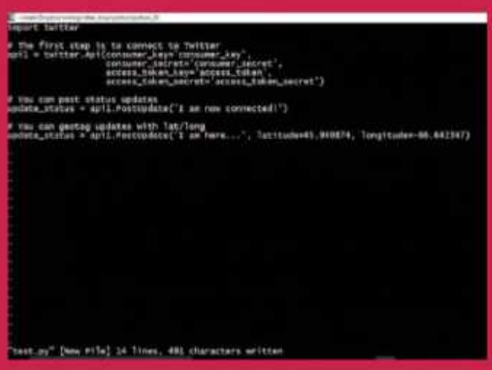
Full code listing

```
import twitter

# The first step is to connect to Twitter
api1 = twitter.Api(consumer_key='consumer_key',
                   consumer_secret='consumer_secret', access_token_
                   key='access_token', access_token_secret='access_
                   token_secret')

# You can post status updates
update_status = api1.PostUpdate('I am now connected!')

# You can geotag updates with lat/long
update_status = api1.PostUpdate('I am here: ', latitude=45.949874,
longitude=-66.642347)

# Posting images taken by your RPi
update_status = api1.PostMedia('This is my picture', 'image1.png')

# Get the last 10 mentions and retweets
mentions = api1.GetMentions(count=10)
retweets = api1.GetRetweetsOfMe(count=10)

# Finding out how many followers you have
followers = api1.GetFollowers()
follower_cnt = len(followers)
```

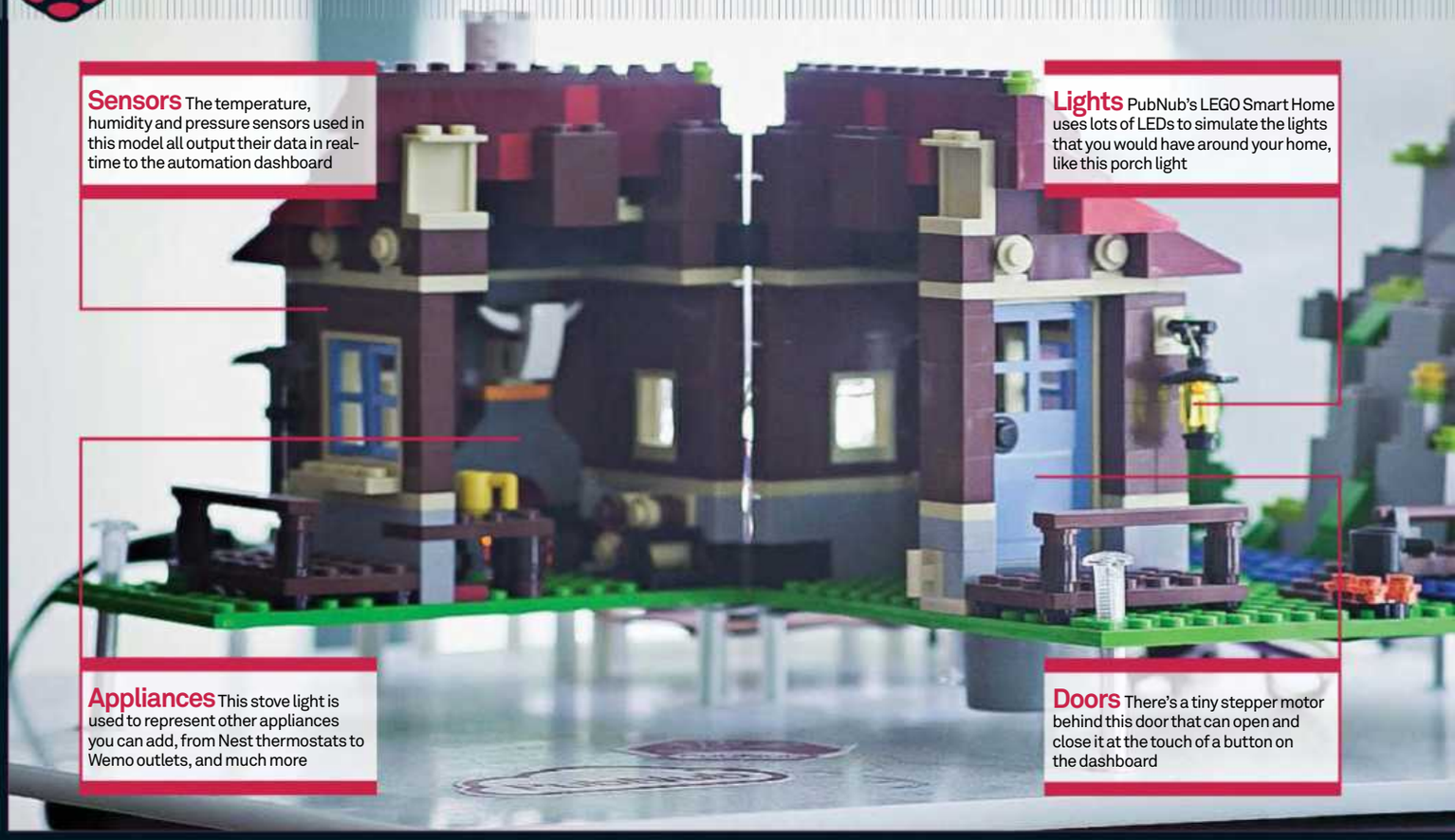


Above Consider setting up a fresh Twitter account specifically for your project

Managing interactions



You might be interested in having more complex control over the Twitter account for your Raspberry Pi project. The Twitter Python module provides full access to the Twitter API, so you can make changes to the relationships between this account and other Twitter users. You can follow another user with the function **CreateFriendship()**. If you change your mind, you can always unfollow a user with the function **DestroyFriendship()**. What might be of more interest, however, is to check out how many people are following you. Or, more specifically, how many people are following your Raspberry Pi project. You can get a sequence of Twitter users who are following your project with the function **GetFollowers()**. The default maximum number of returned followers is 200. So, if your project becomes really popular, you will need to use the function **GetFollowersPaged()** to get all of your followers in groups of 200. You can control which page gets returned with the option **cursor=X**. The defaults also return the statuses for all of these users, which you may not necessarily want to have to process. If so, you can use the option **skip_status=True**. You can also get the number of Twitter friends with the function **GetFriends()**. In this case, the default number of results returned is 20. You can increase this with the option count=X up to a maximum of 200. If you have more than 200 friends, you can get paged results here too. You can control which page of results to return with the cursor option, just as above. If you want to get a broader look at who is checking out your project, you can use the function **GetMentions()** to see who is mentioning the user account for your project. You get the last 20 mentions by default, but you can request up to 200. Another measure of your reach into the public sphere is how many retweets you get. You can find out by using the function **GetRetweetsOfMe()**. The default number returned is 20, but you can ask for up to 100 of the latest retweets. With these extra functions, you can actually have your Raspberry Pi project interacting with the world in general and monitor its own ability to communicate with the public. This might be handy for citizen science projects.



LEGO Smart Home

Bhavana Srinivas and Geremy Cohen from PubNub show us how you can connect together Internet of Things devices as easily as blocks of LEGO

So tell us about PubNub and your LEGO Smart Home.

PubNub is a global data stream network. What this means is that, using the PubNub software, you can talk between devices in real-time. And, going beyond that, we provide an infrastructure as well, so you can scale and build real-time applications. The simplest case is when you have two mobile phones talking to each other – using any kind of instant messaging or a commercial app that requires stocks to be updated in real-time, for example, or any kind of home automation – that kind of real-time messaging is brought to you PubNub. Since we have 14 data centres all over the world, your data gets replicated and we're able to communicate between devices in less than a quarter of a second.

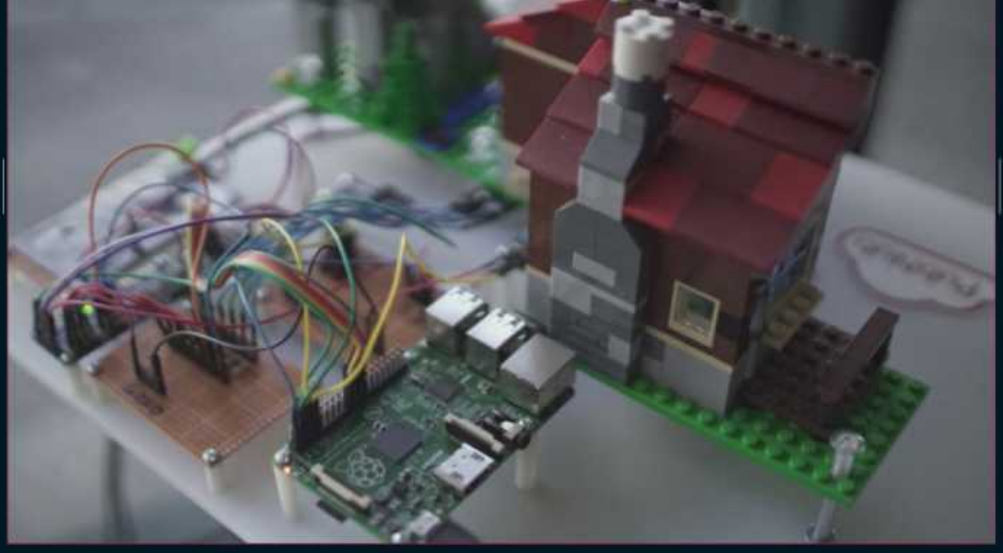
The Internet of Things was picking up at PubNub last year, so we decided to have some cool demos to show how you can integrate PubNub as a software for any kind of home automation that you build out. We were building two projects at the same time – one with the Arduino Uno (bit.ly/11SybiR) and the other with the Raspberry Pi, both to show home automation itself. So with this project, the Raspberry Pi Model B+ would be the brain behind the whole project, powering the different embedded sensors, the stepper motor to control the door, and we used PubNub as a glue to talk between these different devices.

I didn't build this myself – it was actually another engineer here called Geremy who built it over Christmas. We were going to have the software component, which is PubNub, and then on a mobile phone you could control all the devices in your house; the devices being the seven embedded LEDs – used to simulate things like a stove, a fireplace, a barbecue station and lights within the house – and then we had a stepper motor to open and close the door, and a couple of sensors like temperature, humidity, etc, to measure those different values within the house.

So what you can do with this demo is open this dashboard on your phone, typically an app, and the moment you instruct it to switch on the light or open the door, it will reflect immediately on the house. You can get the state of the house continuously, in terms of the voltage or the temperature, the pressure, so that's a typical home automation system.

Components list

- Raspberry Pi B+
- LEDs
- DHT11 temperature/ humidity sensor
- Stepper motor
- I2C bus



Above In principle, this is basically the setup you'd need for an automation setup for your own house

Left If you're feeling creative, why not re-create the smart home model using your own LEGO sets –PubNub-powered Death Star, anyone?

Right The Smart Home model's dashboard enables you to control the LEDs' brightness using sliders

How does messaging between different PubNub-connected devices work?

So PubNub provides real-time communication between any two devices. What happens is that one device is publishing on a particular channel and then another device is subscribing to that same channel, and that's how the message is transmitted from the first device to the second. All you need with PubNub is the publisher key, the subscriber key and then the channel name, and when two devices are using the same set of parameters – the same pub and sub keys – and when one publisher is on that channel, the other receives it on the same channel. On a very basic level, that's how PubNub works.

How much of the software setup was specific to this project – was there a lot of work involved to get PubNub talking to the Pi?

So how it works with PubNub is that we have 70+ SDKs, which means we support that many platforms or devices – so if you're using an iOS device then you can use our iOS SDK, or there's our Python and Java SDKs, and so on. For the Raspberry Pi we chose to use our Python SDK. PubNub provides very easy-to-use APIs, so if you wanted to send out a message then it would be as simple as **pubnub.publish** and to receive a message it would be **pubnub.subscribe**, so the SDKs are already built out, but there was a little bit of tweaking for it to fit this particular home automation model.

Geremy had to build out two parts: the phone or browser app, which is the dashboard that you see, and then the Python scripts running on the Raspberry Pi itself. So for the dashboard he typically used JavaScript and then Android or iOS, depending on the phone application, and then for the Raspberry Pi itself he used Python – Jeremy just had to write a couple of scripts that basically said, 'When I receive a JSON message from my phone saying to switch on LED 1, do so at source'. So the logic for the home automation had to be written in Python, but the software and the documentation already exists.

How easy would it be for people to recreate your LEGO Smart Home model for themselves?

It's super easy. We are actually creating tutorials as well – we're done with the temperature sensor, the humidity sensor, etc, and we just have a little bit of tweaking to do for the lights – so they're going to be smaller follow-ups to the same video and blog that we've posted already. So it's super easy to create and is meant for the hacker community, like a weekend or holiday project where you can just sit and build this whole thing in a couple of days. There's a free tier in PubNub where we provide a sandbox account you can use for this, wherein you're given publish and subscribe keys and you can go crazy. There's a limit on the number of messages you can send but that's way more than what you would need for a free tier. Everything's open, so you can see it on our website, and it's free up to 20 devices.

And how straightforward would it be to scale up this project and use it for real home automation, with things like Philips Hues and Sonos speakers?

We have an intern who recently built out the Hue light bulbs with the Raspberry Pi in about half an hour; it wasn't a big deal at all. PubNub is just plug-and-play – you build out all this software, put it on your hardware, and you don't have to worry about which network you're on, you don't have to configure the routers or firewalls, any of that stuff. It's very easy to scale with PubNub itself because we're global – even if you get up to millions of users, irrespective of where you are in the world you still receive the message in less than a quarter of a second.

Like it?

If you're interested in seeing what else you can do using PubNub and your Raspberry Pi, check out the other tutorials on their website (including a great 101 to get you kick-started): **pubnub.com/blog/tag/raspberry-pi**

Further reading

PubNub recently released an open source JavaScript framework for displaying real-time data in charts, maps and dashboards, which produces gorgeous results and can easily hook into your home automation setup: **pubnub.com/developers/eon**

Study environmental science with a Sensly HAT

Conduct experiments, monitor pollutants and more with this clever little module for your Raspberry Pi

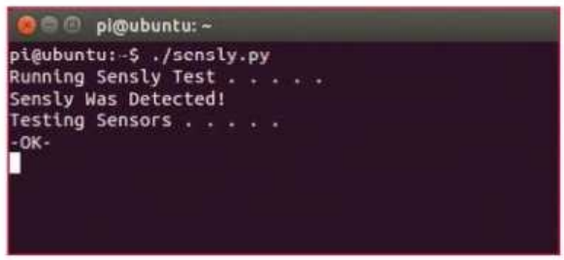
The Sensly is a smart module attached to the Raspberry Pi. It has its own microprocessor that can handle analog data and handle sampling from various sensors without the need to waste your Raspberry Pi's processing power. There are three different gas sensors attached to the board, allowing you to sense a multitude of gases, as well as humidity and temperature sensors. Plus, this Raspberry Pi HAT can be easily extended with its array of analog ports and an I2C interface.

In keeping with the Raspberry Pi way of doing things, the module also provides a Python API that does all of the hard work for you. All of the maths, calibration and error correction is handled by the API itself, allowing you to play around with meaningful data immediately.

01 Setting up your Sensly HAT

This tutorial assumes you have set up your Raspberry Pi and it is connected to the Internet. So, the first step is to plug your Sensly HAT into the Raspberry Pi header. Once you have done this, power on the Raspberry Pi and type the following commands into the terminal:

```
# Install git so we can download the API
sudo apt-get install git
git clone https://github.com/Altitude-Tech/SenslyPi.git
cd SenslyPi
# Install the python API
sudo python setup.py install
```



02 Testing the Sensly

The first step is ensure the Raspberry Pi can communicate with the Sensly HAT and make sure it is functioning correctly. Type the following command and it will respond with "ok", and give a report if everything is working:

```
# Run test sequence on the Sensly
sensly-test
```

What you'll need

- Sensly HAT
  sensly.uk

03 Using the API

We are now going to use a terminal-based text editor to write some code – we're going to use Nano but you can use any editor you like. The following command will create a file called sensly.py and open it for editing.

```
nano sensly.py
```

You can type the following program to get basic data from the Sensly.

```
import os
from sensly import Gases

atmosphere = Gases()

while True:
        print("Humidity Level:")
        print( str(atmosphere.humidity()) )
        print("Temperature:")
        print( str(atmosphere.temp()) )
        time.sleep(5)
```

04 Getting started with gas sensors

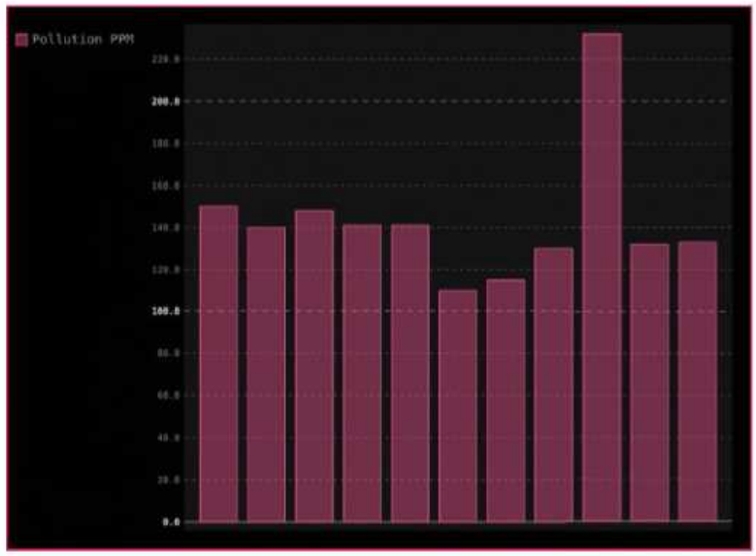
To start using the sensors, it gets a little bit more complicated as we need to preheat the sensors to ensure the readings from the Sensly are stable. To do this we use a loop in the code which waits until the preheating sequence has finished. This takes two to three minutes if you have only just powered up the Sensly.

```
import os
from sensly import Sensly

sensly = Sensly()

# Wait until preheating has finished
# so we can get stable readings
While (sensly.preheated() == False):
        print("Preheating . . . . .")
        time.sleep(1)

print("Sensly is ready to read pollution levels!")
```

05 Getting pollution data

Now that we know how to get the gas sensors ready to read data, it's time to start taking some measurements. Add the following code to the previous program:

```
While(True):
        # Print the general level of pollution
        print("General pollution level (PPM)")
        print( str( atmosphere.pollution()) )

        # Print the level of industrial pollution
        # (e.g. benzene, nitrogen oxides)
        print("Industrial Pollution (PPM)")
        print( str( atmosphere.pollution.industrial()) )

        # Print the Humidity
        print("Humidity")
        print( str( atmosphere.humidity()) )

        time.sleep(20)
```



Above Pygal is a dynamic SVG charting library that offers Python/CSS styling by means of preset themes

06 Adding custom sensors

The Sensly provides five ports for additional sensors, which can easily be controlled with a simple Python script. They are particularly useful because the Raspberry Pi does not provide any native analog ports. This allows you to add extra sensors to suit your projects, such as an LDR to measure light pollution. Using the following script you can easily read analog and digital data from the expansion ports:

```
import sensly
from sensly import Port

sensor1 = Port(Sensly.port1)

print( str( sensor1.analog_read()) )
```

07 Visualising your data

For this we are going to use a great open source project called pygal, which makes it easy to generate cool vector graphs from your data. First of all we need to install pygal, and then we can write some code to generate some information on pollution levels. Run the following commands to set up pygal:

```
sudo apt-get install python-setuptools
sudo easy_install pygal
```

The following code will sample the air for ten hours and generate a vector graph, although you can edit the timings to suit your needs:

```
import pygal
import os
from sensly import Gases

atmosphere = Gases()
graph = pygal.Bar()

samples = []
for i in range(10):
        samples.append(atmosphere.pollution.industrial())

graph.add('pollution', samples)
graph.render_to_file('pollution.svg') # Generate graph
```

Unlocking the ARM cortex

The brain of this add-on board is its ARM cortex, which handles all of the analogue signals, data sampling and communications to the Pi. This is a powerful little chip with many features and its firmware is open source. As a result, you can flash reprogram the HAT from your Raspberry Pi, giving you the flexibility of a microcontroller like the Arduino but with all the advantages of a full operating system – making this a very fun little HAT to tinker about with.

Build a Raspberry Pi Minecraft console

Create a full-functional, Pi-powered games console that you can play Minecraft on and learn how to program too

***Minecraft* means many things to many people, and to Raspberry Pi users it's supposed to mean education.** Not everyone knows, though, that you can still have fun and play *Minecraft* as you normally would.

Using Raspberry Pi, it is also the cheapest way to get a fully-functional version of *Minecraft* up onto your TV. However, in its normal state, just being on a TV isn't the end of it. Using all the features and functions of the Pi, we can take it to a state more fitting of a TV by making it into a hackable, moddable *Minecraft* console.

In this tutorial, we will show you how to set it up in terms of both software and hardware, how to add a game controller to make it a bit better for TV use, and we'll even give you some example code on how to mod it. Now, it's time to get building, so head to Step 1.

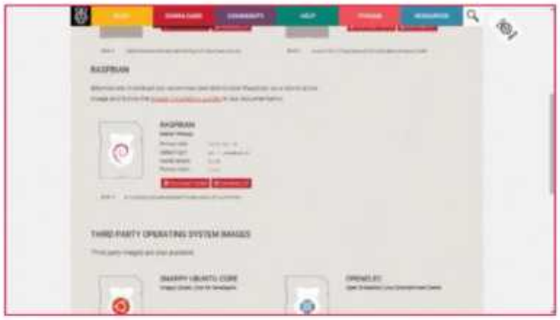
What you'll need

- Raspberry Pi 2
- Latest Raspbian image
  raspberrypi.org/downloads
- Minecraft Pi Edition
  pi.minecraft.net
- Raspberry Pi case
- USB game controller (PS3 preferable)

Above Give *Minecraft: Pi Edition* a quick test before you start building the console

01 Choose your Raspberry Pi

Before we start anything, everything we plan to do in this tutorial will work on all Raspberry Pi Model Bs with at least 512 MB of RAM. However, *Minecraft: Pi Edition* can chug a little on the original Model Bs, so we suggest getting a Raspberry Pi 2 to get the most out of this tutorial.



02 Prepare your Raspberry Pi

Minecraft: Pi Edition currently works on Raspbian. We recommend you install a fresh version of Raspbian, but if you already have an SD card with it on, the very least you should do is:

```
sudo apt-get update && sudo apt-get upgrade
```

03 Prepare Minecraft

If you've installed Raspbian from scratch, *Minecraft* is actually already installed – go to the Menu and look under Games to find it there ready. If you've just updated your version of Raspbian, you can install it from the repos with:

```
$ sudo apt-get install minecraft-pi
```

04 Test it out

If you've had to install *Minecraft*, it's best just to check that it works first. Launch the desktop, if you're not already in it, with **startx** and start *Minecraft* from the Menu. *Minecraft: Pi Edition* is quite limited in what it lets you do, but it does make room for modding.

05 X setup

If you have a fresh Raspbian install and/or you have your install launch into the command line, you need to set it to load into the desktop. If you're still in the desktop, open up the terminal and type in **raspi-config**. Go to Enable Boot to Desktop and choose Desktop.

> If you've installed Raspbian from scratch, Minecraft is actually already installed – go to the Menu and look under Games to find it

06 Set up Python

While we're doing set up bits, we might as well modify *Mincecraft* using Python for a later part of the tutorial. Open up the terminal and use:

```
$ cp /opt/minecraft-pi/api/python/mcpi ~/minecraft/
```

07 Minecraft at startup

For this to work as a console, we'll need it to launch into *Minecraft* when it turns on. We can make it autostart by going into the terminal and opening the autostart options by typing:

```
$ sudo nano /etc/xdg/lxsession/LXDE-pi/autostart
```

08 Autostart language

In here, you just need to add **@minecraft-pi** on the bottom line, save it and reboot to make sure it works. This is a good thing to know if you also want other programs to launch as part of the boot-up process.

09 Turn off

For now, we can use the mouse and keyboard to shut down the Pi in the normal way, but in the future you'll have to start turning it off by physically removing power. As long as you've exited the *Minecraft* world and saved, that should be fine.

Updates to Pi Edition?

Minecraft: Pi Edition hasn't received an update for a little while, but it was previously limited by the original Model B. Now with more power, there may be an update that adds more to it, but right now there's no indication of that. If it does come though, all you need to do is update your Pi with: **sudo apt-get update && sudo apt-get upgrade**.

Getting power to the Raspberry Pi 2 so that it runs properly can be tricky if you're using a USB port

3D-print a case

Aaron Hicks at Solid Technologies designed this *Minecraft* case for the Raspberry Pi and shared it on GrabCAD. We've uploaded our slightly modified version to **FileSilo.co.uk** along with your tutorial files for this issue. All you need to do is send the STL file to a 3D printing service – many high street printing shops have at least a MakerBot these days – and they will 3D-print the case for you. It should only cost around £15.

10 The correct case

In this scenario, we're hooking this Raspberry Pi up to a TV, which means it needs a case so that there's less chance of damage to the components from dust or static. There are many good cases you can get – we are using the Pimoroni Pibow here as you can mount it to the back of the TV. Alternatively, you could get really creative and 3D-print your own case, as you can see on page 58. Check out the boxout just to the left.

11 Find the right power supply

Getting power to the Raspberry Pi 2 so that it runs properly can be tricky if you're using a USB port or a mobile phone charger – the former will be underpowered and the latter is not always powerful enough. Make sure you get a 2A supply, like the official Raspberry Pi one.

12 Go wireless

We understand that not everyone has an ethernet cable near their TV, so it may be a good idea to invest in a Wi-Fi adapter instead. There is a great list of compatible Wi-Fi adapters on the eLinux wiki: **elinux.org/RPi_VerifiedPeripherals**.

13 Mouse and keyboard

Now that we have the Raspberry Pi ready to be hooked up, you should look at your controller situation – do you want to be limited by the wires or should you get a wireless solution instead? We will cover controller solutions over the page, but it's worth considering now.

14 Get ready for SSH

It will be easier to create and apply scripts to *Minecraft* by uploading them via the network rather than doing it straight on the Pi. In the terminal, find out what the IP address is by using **ifconfig**, and then you can access the Pi in the terminal of another networked computer using the following:

```
ssh pi@[IP address]
```

15 Have a play

At this stage, what we have built is a fully-functional *Minecraft* console. Now, at this point you could start playing if you so wish and you don't need to add a controller. You can flip over to page 62 now if you want to begin learning how to mod your *Minecraft* and do a bit more with it to suit your needs. However, if you do want to add controller support then carry on and take a look at Step 16.

Controls

Here's the full layout of the buttons used by the PS3 controller by default – you can change them in the script that you download in Step 18

16 Add controller support

Make sure the controller input functions are installed on the Raspberry Pi. To do this, **ssh** into the Raspberry Pi like we did in Step 14 (where 'raspberry' is the password) and install the following package:

```
$ sudo apt-get install xserver-xorg-input-joystick
```

17 Controller mapping

We have a controller map for the PS3 controller that you can download straight to your Pi, and with a bit of tweaking can fit most USB controllers as well. Go to the controller configuration folder with:

```
$ cd /usr/share/X11/xorg.conf.d/
```

18 Replace the controller mapping

We'll remove the current joystick controls by using `sudo rm 50-joystick.conf` and then replace by downloading a custom configuration using:

```
$ sudo wget http://www.linuxuser.co.uk/wp-content/
uploads/2015/04/50-joystick.conf
```

19 Reboot to use

After a reboot to make sure everything's working, you should be able to control the mouse input on the console. R2 and L2 are the normal mouse clicks and can be used to navigate the *Minecraft* menu to access the game.

20 Go full-screen

So far you may have noticed that *Minecraft* is running in a window – you can click the full-screen button to make it fill the screen, however you then heavily limit your mouse control. Thanks to the controller, you can get around that. As soon as you load the game, make sure you use the sticks for movement and the d-pad for selecting items in the inventory.

Xbox controllers

Unfortunately, Xbox 360 controllers work slightly differently with Linux. As they use their own drivers that are separate to the normal joystick drivers we used for the PS3 pad and other USB controllers, a 360 controller doesn't work as a mouse and is harder to assign specific functions to. This makes it tricky to use in a situation such as this.

Mod your Minecraft

Here is some example code, and explanations for it, so that you can learn how to program in Python and mod Minecraft Pi

We program *Minecraft* to react in python using the API that comes with *Minecraft: Pi Edition* – it's what we moved to the home folder earlier on. Now's a good time to test it – we can do this remotely via SSH. Just **cd** into the Minecraft folder in the home directory we made, and use **nano test.py** to create our test file. Add the following:

```
from mcpi.minecraft import Minecraft
from mcpi import block
from mcpi.vec3 import Vec3
mc = Minecraft.create()
mc.postToChat("Hello, Minecraft!")
```

Save it, and then run it with:

```
$ python test.py
```

"Hello, Minecraft!" should pop up on-screen. The code imports the Minecraft function from the files we moved earlier, which allows us to actually use Python to interact with *Minecraft*, along with the various other functions and modules imported. We then create the mc instance that will allow us to actually post to *Minecraft* using the postToChat function. There are many ways you can interact with *Minecraft* in this way – placing blocks that follow the player, creating entire structures and giving them random properties as they're spawned as well. There are very few limits to what you can do with the Python code, and you can check out more projects here: **https://mcpipy.wordpress.com**.

Over the page, we have a full listing for a hide and seek game that expands on the kind of code we're using here, where the player must find a diamond hidden in the level, with the game telling you whether you're hotter or colder. You can write it out from scratch or download it to your Pi using the following commands:

```
$ wget http://www.linuxuser.co.uk/tutorialfiles/
Issue134/ProgramMinecraftPi.zip
$ unzip ProgramMinecraftPi.zip
$ cp Program\ MinecraftPi/hide_and_Seek.py ~/minecraft
```

Check out the annotations to the right to see how it works.

> We program Minecraft to react in Python using the API that comes with Minecraft Pi – it's what we moved to the home folder earlier

Below You can see the hidden diamond just to the left of the crosshair at the centre of this screenshot

Full code listing

Import

Here we're importing the necessary modules and APIs to program *Minecraft*. Most importantly are the files in the mcpi folder that we copied earlier

```
from mcpi.minecraft import Minecraft
from mcpi import block
from mcpi.vec3 import Vec3
from time import sleep, time
import random, math
```

Locate

We connect to *Minecraft* with the first line, and then we find the player's position and round it up to an integer

```
mc = Minecraft.create()
playerPos = mc.player.getPos()

def roundVec3(vec3):
  return Vec3(int(vec3.x), int(vec3.y), int(vec3.z))
```

Range finding

Calculate the distance between the player and diamond. This is done in intervals later on in the code, and just compares the co-ordinates of the positions together

```
def distanceBetweenPoints(point1, point2):
  xd = point2.x - point1.x
  yd = point2.y - point1.y
  zd = point2.z - point1.z
  return math.sqrt((xd*xd) + (yd*yd) + (zd*zd))
```

Creation

Create a random position for the diamond within 50 blocks of the player position that was found earlier

```
def random_block():
  randomBlockPos = roundVec3(playerPos)
  randomBlockPos.x = random.randrange(randomBlockPos.x - 50, randomBlockPos.x + 50)
  randomBlockPos.y = random.randrange(randomBlockPos.y - 5, randomBlockPos.y + 5)
  randomBlockPos.z = random.randrange(randomBlockPos.z - 50, randomBlockPos.z + 50)
  return randomBlockPos
```

Start

This is the main loop that actually starts the game. It asks to get the position of the player to start each loop

```
def main():
  global lastPlayerPos, playerPos
  seeking = True
  lastPlayerPos = playerPos
```

Notification

This part sets the block in the environment and pushes a message using postToChat to the *Minecraft* instance to let the player know that the mini-game has started

```
  randomBlockPos = random_block()
  mc.setBlock(randomBlockPos, block.DIAMOND_BLOCK)
  mc.postToChat("A diamond has been hidden - go find!")
```

Checking

We start timing the player with timeStarted, and set the last distance between the player and the block. Now we begin the massive while loop that checks the distance between the changing player position and the fixed diamond. If the player is within two blocks of the diamond, it means they have found the block and it ends the loop

```
  lastDistanceFromBlock = distanceBetweenPoints(randomBlockPos, lastPlayerPos)
  timeStarted = time()
  while seeking:

    playerPos = mc.player.getPos()

    if lastPlayerPos != playerPos:
      distanceFromBlock = distanceBetweenPoints(randomBlockPos, playerPos)
      if distanceFromBlock < 2:
        seeking = False
```

Message writing

If you're two or more blocks away from the diamond, it will tell you whether you're nearer or farther away than your last position check. It does this by comparing the last and new position distance – if it's the same, a quirk in Python means it says you're colder. Once it's done this, it saves your current position as the last position

```
      else:
        if distanceFromBlock < lastDistanceFromBlock:
          mc.postToChat("Warmer " + str(int(distanceFromBlock)) + " blocks away")
        if distanceFromBlock > lastDistanceFromBlock:
          mc.postToChat("Colder " + str(int(distanceFromBlock)) + " blocks away")

      lastDistanceFromBlock = distanceFromBlock

    sleep(2)
```

Success

It takes a two-second break before updating the next position using the sleep function. If the loop has been broken, it tallies up your time and lets you know how long it was before you found the diamond. Finally, the last bit then tells Python to start the script at the main function

```
  timeTaken = time() - timeStarted
  mc.postToChat("Well done - " + str(int(timeTaken)) + " seconds to find the diamond")

if __name__ == "__main__":
  main()
```

Minecraft blocks The NFC tags within have instructions on them. When scanned against the PN532, they pass them on to the *Minecraft* API

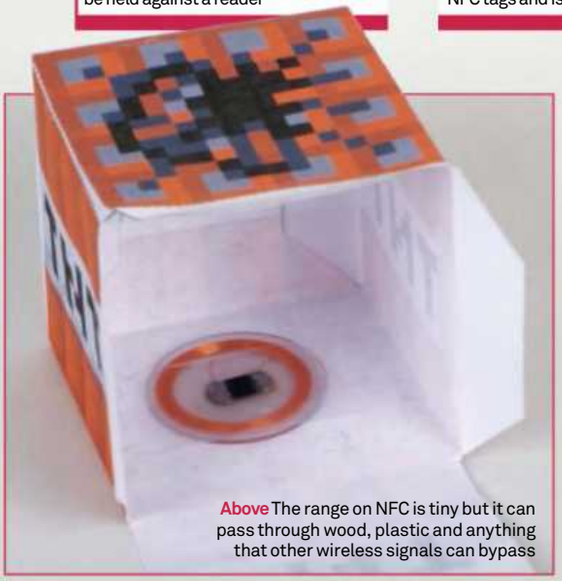
Components list

- Raspberry Pi 2
- Raspberry Pi GPIO Breakout
- PN532 RFID/NFC Breakout
- Breadboard
- MIFARE Classic 1K cards
- Printed and folded Minecraft cubes

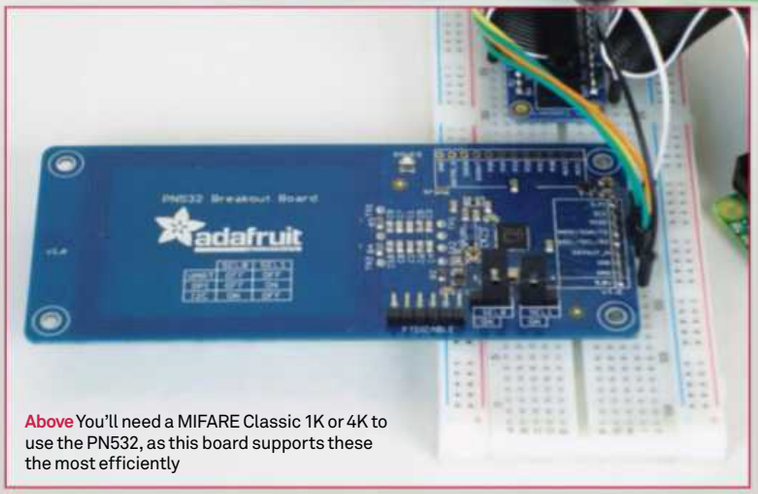
Raspberry Pi Tony is using the Adafruit Pi Cobbler Plus in this project, which breaks out the GPIO, SPI and I2C pins for use with the PN532

NFC tags NFC tags are designed for use at close range, usually inside other objects or even stickers that can be held against a reader

PN532 NFC Breakout Costing $40, the PN532 breakout is versatile. It works with RFID tags and NFC tags and is supported by libnfc



Above The range on NFC is tiny but it can pass through wood, plastic and anything that other wireless signals can bypass



Above You'll need a MIFARE Classic 1K or 4K to use the PN532, as this board supports these the most efficiently

Minecraft NFC

Inpired by Amiibos, Tony DiCola makes NFC-enabled papercraft blocks that can be used to build inside Minecraft

Why did you decide to start working with NFC – what was it that first interested you?

I think it was a neat way to explore using physical things with a game – taking a device or an object and making it more interactable. I was a little bit inspired by Amiibos and Skylanders – those little figures you can buy that have NFC in them. So I was thinking, what can I do to make something myself out of this? What's a DIY thing to try? And then I realised that the Raspberry Pi has *Minecraft* and the cool thing about *Minecraft Pi* is that it has got a whole little Python API, so it's really easy to use. With a few lines of Python you can create blocks and mess around in the *Minecraft* world, so I thought that was perfect – I could just put both of these together and make some sort of physical thing to interact with the *Minecraft* world.

So how exactly does it work?

Basically, it has a little NFC reader, which is this circuit board that connects to the Raspberry Pi and uses an SPI connection (a typical connection for embedded devices), and then the Pi has the GPIO ports, so you can get really good low-level access to devices with that. NFC is a wireless Near Field Communication protocol, so it's kind of cool in that you can have these little passive devices, like a tag or a card, and you just hold them near the antenna on the RFID reader and it'll energise them, sending a few little bits of data between them. So the NFC reader is connected to the Pi and there's some Python code that I wrote – a little library – to interact with the NFC reader. There's a little program that you run (also in Python) that basically talks to the NFC reader, and it's in a passive listening mode, so it just waits to see when a card is swiped. Ahead of time, you build these little blocks and put NFC cards inside them and then you scan those – each one has a unique ID associated with it, so once you get the ID then you can have a little configuration that says 'this ID equals the wood block', 'this ID equals the TNT block', etc. So the program just waits to see a block that's swiped and then, using the *Minecraft* API, it tells *Minecraft* to create a new block wherever the player is standing now.

So you wrote the library for the Adafruit PN532 RFID breakout that you are using in this project?

Exactly – the Python port of it. There's also an Arduino port and it's pretty standard, a typical kind of embedded code. The nice thing with this breakout is that you don't really have to talk at a super-low level with the different NFC protocols – the chip on here takes care of that for you, so really the library is just talking to the chip and saying, 'run the listening command' or 'run the read or write command'. It abstracts away how to initialise the device and it has a bit of a framing protocol – when you send a command it has a hash and other stuff associated with it to make sure it gets the correct data, so the library takes care of all that for you. You can just look at the datasheet and it tells you all the different commands that it supports, and in order to send those commands you have to frame them in the right way, but the library will do that for you. And even above that there are some higher-level functions that just, for example, wait for a card, and then once you have a card there's a function that says 'give me all the data from the card', so it's meant to be real simple and easy to use, really make you able to be productive with it.

> I was a little bit inspired by Amiibos and Skylanders – those figures you can buy that have NFC in them

How much information can you store in these NFC tags? Could you write the layout for a *Minecraft* building on a chip?

The cards that I was using, the MIFARE Classics, store 1KB of data and I'm not actually even storing much data – it's just the type of block, a byte or two. But you could actually create some arrangement of blocks, like a pyramid or a house, up to a kilobyte, so you'd have some restrictions in that each block will need a position. You'll probably need three or four bytes for that (or maybe even three or four 16- or 32-bit values), so it could probably store maybe 100 blocks or so in an arrangement on a 1KB card. But there are bigger cards – I think there are 4KB ones. So it could be a cool thing to support for the future, extend it to support a structure or something that you make ahead of time – the tricky thing is that you'd need some kind of editor, ideally. *Minecraft* on the Pi is pretty basic, so you would have a bit of a tough time trying to define the structure without just dropping someone down to configuration and saying, 'Okay, write out the exact position of the blocks'. But I guess if someone were really ambitious, they could make a little 3D editor or something to place the blocks.

Have you experimented using the PN532 in any Raspberry Pi projects outside of *Minecraft* up to now?

Not too much yet. I was kind of curious to see if we could read Amiibos – what do they actually store on the cards? – and you can actually read them. There's a community on Reddit trying to hack the Amiibos and actually figure out what the data is, but it's all encrypted by Nintendo so you can't really understand it. I was hoping that maybe you could actually clone an Amiibo but the 532 unfortunately doesn't support writing to the Amiibo; they use special tags – NTAG216, I think – and it's a weird format, quite hard to find. Nintendo didn't want people to copy them, obviously!

Like it?

Check out Tony's tutorial on the Adafruit Learning System if you're up for having a go yourself: **bit.ly/1d9oqSh**. Also give the PN532 guide that's recommended on the intro page a read through, as it's very useful (written by Ladyada).

Further reading

Up for something a little more challenging? The University of Cambridge's Computer Laboratory has an excellent tutorial on making your own NFC coil and then setting up communication protocols for it: **bit.ly/1FXYh4g**.

Below The Raspberry Pi itself sits just beside the vivarium, processing data from the DHT22s

Components list

- Raspberry Pi Model B+
- 3 DHT22 temperature-humidity sensors
- 4 relay modules
- Outlet boxes
- LEDs
- 2 Mini 5V blower fans
- 2 2N222 transistors
- 2 10K resistors
- 10Ω, 0.5W resistor

RasPiViv

Nate Bensing tells us how he built an environmental control system to keep seven poison dart frogs cosy

So, what do you keep in the vivarium?
Right now I have seven poison dart frogs – they're frogs from South America that, in the wild, excrete poison alkaloids, but in captivity, because their diet is just fruit flies, they can't produce any poison. They're something I've been interested in for quite a long time. I think I saw them first when I was in grade school at a trip to the Denver zoo, and I just thought they were the coolest animals I'd ever seen in my life. I've wanted to keep them since then but the opportunity never came up – they're kinda rare – until I found a breeder on Craigslist who has an incredible collection and he breeds them to support his hobby. So right now I have a total of seven: two blue poison dart frogs, which I think could be a breeding pair, and I recently obtained five yellow-banded poison dart frogs.

What kind of requirements do you have for the vivarium, then?
The temperature is really important, which a lot of people have trouble with because your house temperature is going to be about the same as an enclosed box, give or take, as lighting can heat things up. But you have to maintain specific temperatures between about 75 and 85 degrees, and the humidity is even more important because the frogs use the humidity to thermoregulate, kinda like how we sweat.

So basically, what I needed was a way to monitor and regulate the humidity. I looked around online for a couple of days trying to come up with a solution – a lot of people use little timers, and there are a couple of systems that are made for this but they don't do very much.

What hardware did you use to make your own solution?
Well, the Raspberry Pi is set up as a LAMP server. I started playing around with the DHT22 temperature-humidity sensor – originally I just wanted to monitor that, the temperature and the humidity, but then I got into using relays to control the lighting and it just progressed further and further. I was just making this for myself and I posted on a forum. Then a lot of people seemed really interested, so I said I'd clean things up and throw together a guide (see **www.raspiviv.com**).

So the temperature and humidity sensors are read every 60 seconds and logged into a database. Using WiringPi and Adafruit's DHT22 sensor (and the driver for it, which is brilliant and works really well), lighting is controlled by regular relays or relay modules, and the fan is just a basic transistor switch. The main idea behind the fan is that if the atmosphere is too saturated with water then the frogs can't thermoregulate. So the database is read every five minutes, and when it reaches 95% humidity it then kicks on the fan to blow in some fresh air.

Do you SSH in to control all this or do you have a web interface?
Yeah, there's a whole web interface where you can check out the current readings. Check out the Demo section on my website and the first thing that pops up is my blue poison dart frog vivarium. It gives you the current temperature and humidity, and then also the readings for the last hour. If you click on the icon that looks like a grid of buttons (manual controls), you can manually control your lighting, any misting system or fans you have – essentially, for any component that you want to control, it's just a matter of getting a relay module and wiring it up.

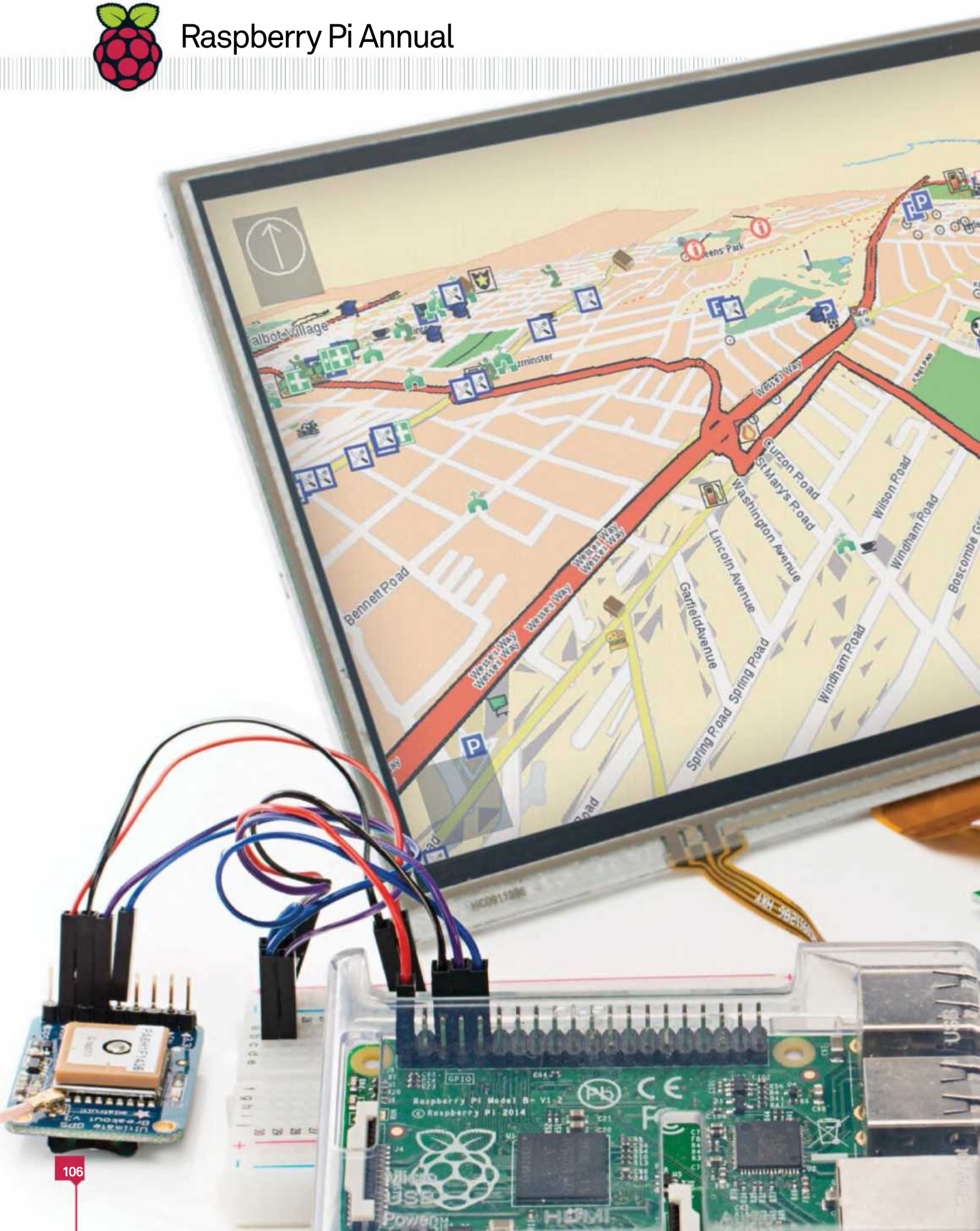
Are you planning to upgrade the RasPiViv software at any point?
Yeah, I am hoping to. I started work on this in my downtime and got insanely busy, so unfortunately I haven't been able to do a lot with it. I'm hoping to get some RF power outlets soon so that, rather than wiring up a little power outlet box, you can just use these prebuilt outlets that you plug into your wall and they give you a little remote control. I am hoping to implement some wireless stuff and I'd definitely like to make it more user friendly, rather than people manually adding cron jobs to things. I'd like them to be able to do it through the browser interface, stuff like that.

What you don't see in the demo is that there's also a login system – you can make an account and keep it secure – so I want to give people that and I'll probably run a tutorial for setting it up. I've been playing around with the Raspberry Pi camera module and hoping to include it, so now we have a Raspberry Pi 2 that is a lot more capable, it could potentially pull off some kind of live camera that you can watch as well as all the other stuff that it does.

Left The RasPiViv web interface shows you the temperature and humidity readings over time

Like it?
Fancy building your own Vivarium controller? Check out the excellent step-by-step guide on Nate's website that takes you from NOOBS to the humidity regulation cron job (**bit.ly/1HTKyeX**).

Further reading
Looking for more inspiration for sensor-driven projects? There are some great ones featured on The Raspberry Pi Foundation blog, like the Feeder Tweeter and the PiPlanter: **bit.ly/1Ak37mu**.


Bennett Road
Wester Way
Wester Way
Wester Way
Wester Way
Wester Way
Garfield Avenue
Lincoln Avenue
Washington Avenue
St Mary's Road
Curzon Road
Wilson Road
Windham Road
Windham Road
Spring Road
Spring Road
Raspberry Pi Model B+ V1.2
© Raspberry Pi 2014

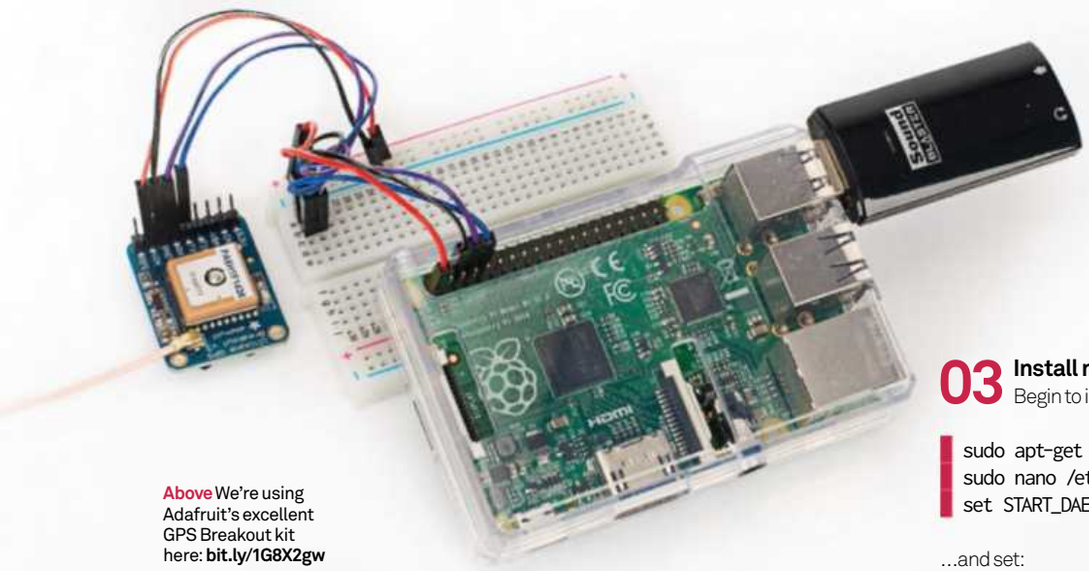
Raspberry Pi Car Computer

Make your own touchscreen navigation system that gives directions, local weather reports and plays music

Cars are getting clever. These days, with smart navigation interfaces built into new cars, you don't need to go out and buy yourself a TomTom to get help with directions. But if you've got a Raspberry Pi then you don't even need to buy that – let alone a new car!

In this project we will show you how to build your own car computer with your Pi, a quality touchscreen like the 9-inch model from SainSmart that we're using here, and a few other bits like a GPS module and USB 3G modem. Your CarPi will be able to use open source navigation software Navit to show your route map on screen, plus speech synthesis to read out directions, and it will also be able to check your location and give you weather reports. It'll work as a music player too, of course.

It's an ambitious project, but you will gain a solid understanding of custom-made interfaces, navigation software and geolocation data, touchscreen calibration, speech synthesis and more. While you don't have to use the same SainSmart screen as us, we do recommend it for this project as it is one of the few large touchscreens out there for the Pi. There are more improvements at the end too, so check the components list, make sure you've got everything and let's get started!

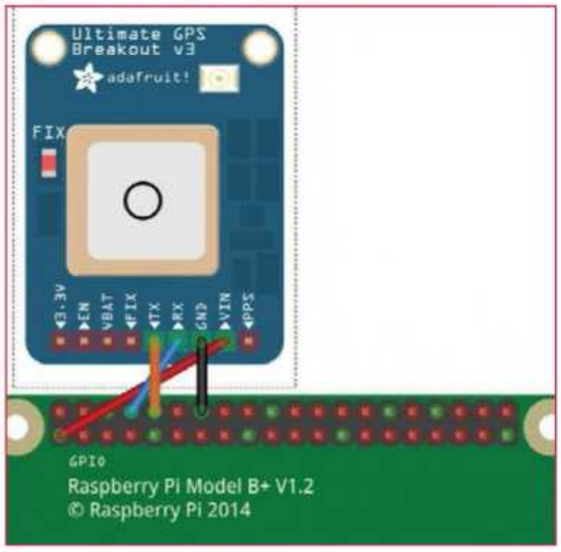


Above We're using Adafruit's excellent GPS Breakout kit here: **bit.ly/1G8X2gw**

01 Basic configuration

Boot up your Raspberry Pi and expand the filesystem using **raspi-config**. Go to Advanced Options and disable the Serial connection – you'll need this to talk to the GPS module later. In raspi-config, enable X at boot as the pi user. Say Yes to reboot. Once rebooted, ensure your packages are up to date with:

```
sudo apt-get update
sudo apt-get upgrade
```



02 Connect GPS module

Solder the pin headers onto the Adafruit GPS module. You can also solder the battery connector which is used to keep the device partially active, giving a faster fix. You only need to use 4 pins: 3.3V, ground, serial transmit and serial receive. Power the Pi off again before connecting anything.

As we are using GPS, the antenna will have to go outside or under a window to gain signal. Connect the antenna to the board and power everything back on. The light on the GPS module will flash frequently while finding a fix. Once it has one, it will blink every 15 seconds.

03 Install navigation software

Begin to install the Navit navigation software by entering:

```
sudo apt-get install navit gpsd gpsd-clients espeak
sudo nano /etc/default/gpsd
set START_DAEMON="true"
```

...and set:

```
DEVICES="/dev/ttyAMA0"
```

Start the GPS daemon with:

```
sudo /etc/init.d/gpsd start
```

You can check it's working by looking at the GPS data with:

```
cgps -s
```

04 Connect the screen

The SainSmart screen doesn't come with any written instructions. Instead there is a YouTube video on their website with details about how to put it together: **bit.ly/1DF6eJJ**. The important part is that the DC power supply should be 12V.

05 Set the screen resolution

We will have to force the correct resolution (1024x600) for the screen by editing /boot/config.txt with sudo. To do so, add the following options:

```
framebuffer_width=1024
framebuffer_height=600
hdmi_force_hotplug=1
hdmi_cvt=1024 600 60 3 0 0 0
hdmi_group=2
hdmi_mode=87
```

For the changes to properly take effect you will need to reboot with `sudo reboot`.

06 Download kernel source

To start the touchscreen, you need to compile an extra kernel module to support it. The program rpi-source (**github.com/notro/rpi-source/wiki**) will find the source of your kernel. Install rpi-source with:

```
sudo wget https://raw.githubusercontent.com/notro/
rpi-source/master/rpi-source -O usr/bin/rpi-source
&& sudo chmod +x /usr/bin/rpi-source && /usr/bin/
rpi-source -q -tag-update
```

Then run `rpi-source` to get the source of the running kernel.

07 Update GCC

Recent Raspberry Pi kernels are compiled with GCC 4.8. Raspbian only comes with 4.6 so you will have to install 4.8 to continue with the following steps. Do this by entering:

```
sudo apt-get install -y gcc-4.8 g++-4.8 ncurses-dev
```

Then you have to set GCC 4.8 as the default:

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.6 20
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 50
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.6 20
sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.8 50
```

08 Pick the module to compile

Rpi-source puts the kernel source in a folder called 'linux'. To choose the USB Touchscreen Driver, enter the following:

```
cd linux
make menuconfig
Device Drivers -> Input device support -> Generic input layer (needed for keyboard, mouse, ...) -> Touchscreens (press space to include) -> USB Touchscreen Driver (press M to make module)
```

Once you've done that, you then need to make sure you save your changes as '.config' and run scripts/diffconfig to see the differences.

09 Compile and install the module

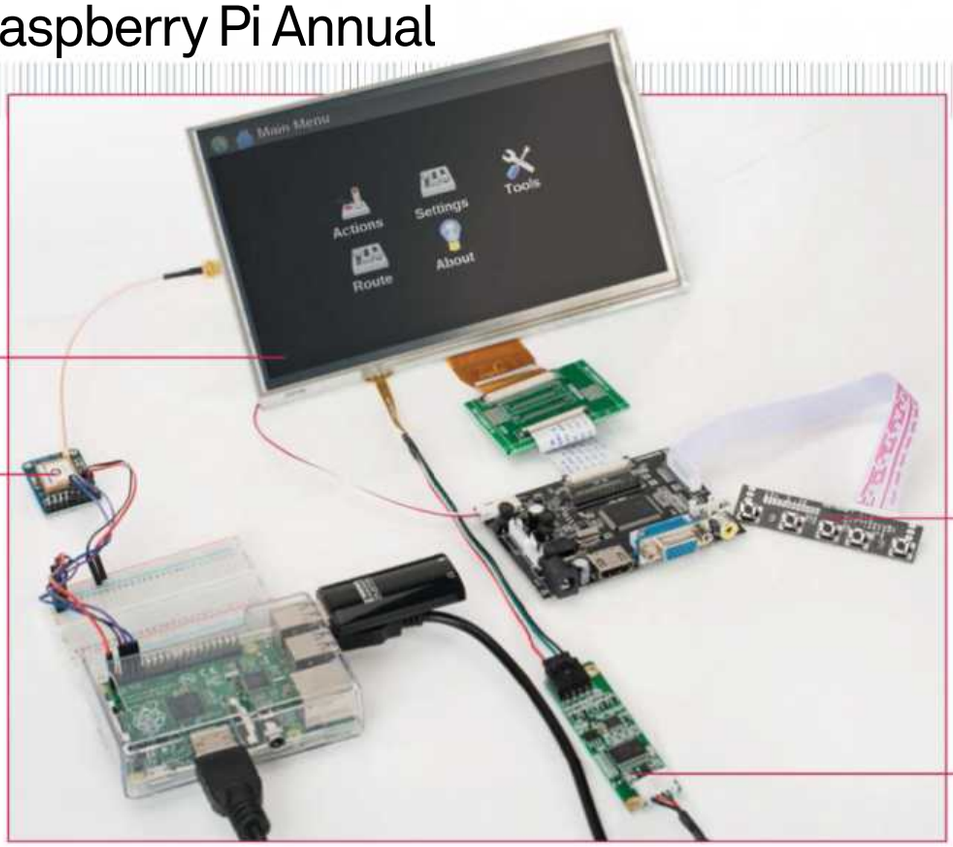
Now you need to compile and install the module. Do so by entering:

```
make prepare
make SUBDIRS=drivers/input/touchscreen modules
sudo make SUBDIRS=drivers/input/touchscreen modules_install
sudo depmod
```

If you unplug and reconnect the touchscreen, it should work fine but it will probably need calibrating.

Full code listing

```
#!/usr/bin/env python2

import os, sys, requests, pygame
from gps import *
from pygame.locals import *

class WeatherClient:
    apikey = "7232a1f6857090f33b9d1c7a74721"

    @staticmethod
    def latlon():
        gpsd = gps(mode=WATCH_ENABLE)

      # Needs better error handling
        try:
            while True:
                report = gpsd.next()
                if report['class'] == 'TPV':
                    gpsd.close()
                    return report['lat'], report['lon']
        except:
            return None, None

    @staticmethod
    def usefuldata(j):
        # Returns a string of useful weather data from a LOT of json
        d = j['data']['current_condition'][0]
        out = "Now - Temp: {0}C, Feels Like: {1}C, Description: {2}\n"\
              .format(d['temp_C'],
                      d['FeelsLikeC'],
                      d['weatherDesc'][0]['value'])

        hourly = j['data']['weather'][0]['hourly']
        hour_count = 1
        for h in hourly:
            out += ("+{0}hr - Temp: {1}C, Feels Like: {2}C, Chance of Rain:"
                    " {3}%, Description: {4}\n")\
                    .format(hour_count,
                            h['tempC'],
                            h['FeelsLikeC'],
                            h['chanceofrain'],
                            h['weatherDesc'][0]['value'])
        hour_count += 1

        # Rstrip removes trailing newline
        return out.rstrip()

    @staticmethod
    def update():
        errstr = "Error getting weather data"

        lat, lon = WeatherClient.latlon()
        if lat == None or lon == None:
            return errstr

        api_req = ("http://api.worldweatheronline.com/free/v2/weather.ashx"
        "?q={0}%2C{1}&format=json&key={2}").format(lat, lon,
                                                  WeatherClient.apikey)
        r = None
```



SainSmart's 9-inch HDMI/VGA touchscreen (**bit.ly/1Ciu4H9**) has a fantastic display and is perfect for all sorts of Pi projects

The screen control panel that comes with the SainSmart screen enables you to easily change the display settings (i.e. brightness, contrast, etc) as well as the input (i.e. HDMI, VGA, AV1, etc)

Adafruit's Ultimate GPS Breakout kit provides Navit and the weather function with the location data that they require

As well as the main controller board, the touch screen is connected to a four-line USB controller which then plugs into the Pi's USB port

10 Calibrate the touchscreen

At this point, you can easily calibrate the touchscreen by entering the following:

```
cd /etc/X11
sudo mkdir xorg.conf.d
cd xorg.conf.d
sudo nano 99-calibration.conf
```

...with the following content:

```
Section "InputClass"
    Identifier "calibration"
    MatchProduct "eGalax Inc. USB TouchController"
    Option    "SwapAxes"    "1"
    Option    "InvertX"     "1"
EndSection
```

Invert X actually inverts Y because the axes have been swapped around. Reboot again for these changes to occur. Now the calibration is roughly correct, download an input calibrator that Adafruit have packaged already.

```
wget http://adafruit-download.s3.amazonaws.com/
xinput-calibrator_0.7.5-1_armhf.deb
sudo dpkg -i xinput-calibrator_0.7.5-1_armhf.deb
DISPLAY=:0.0 xinput_calibrator
```

DISPLAY=:0.0 is useful because you can run the program from any terminal (including an SSH session) and have it appear on the touchscreen. Touch the points on the screen as prompted. Once the program is finished, you should get an output that is similar to the following:

```
Option    "Calibration"    "84 1957 270 1830"
```

Add it to the '99-calibration.conf' file that we created earlier just below the other Option entries.

11 Download maps

Navit needs maps; download them from **maps.navit-project.org**. You can either use the web browser on the Pi or download the map from another machine and copy it using **scp**. Use the predefined area option to select where you live. The smaller the area that you pick, the less data you will have to process. Here the UK has a map size of 608 MB. Now move the map to the navit folder:

```
mkdir -p /home/pi/.navit/maps
mv /home/pi/Downloads/$your_map /home/pi/.
navit/$country.bin
```

For example:

```
mv /home/pi/Downloads/osm_bbox_-9.7,49.6,2.2,61.2.bin
/home/pi/.navit/maps/UK.bin
```

12 Navit configuration

Sudo-edit /etc/navit/navit.xml with your favourite editor. Search for openstreetmaps. Now disable the sample map above, enable the openstreetmap mapset and set the data variable to where you just moved your map. In this case it looks like this:

```
<!-- Mapset template for openstreetmaps -->
<mapset enabled="yes">
<map type="binfile" enabled="yes" data="/home/
  pi/.navit/maps/UK.bin"/>
</mapset>
```

Then search for osd entries similar to:

```
<osd enabled="yes" type="compass"/>
```

...and enable the ones you want – we recommend enabling them all. You may want to zoom in closer than the default map layout. A zoom value of 64 is useful.

Embed the screen

We've looked at the PiTFT and the HDMIPi before, but the SainSmart touchscreen we're using here is uniquely suited to many embedded projects. It's larger than the PiTFT but also without the large bezels of the HDMIPi – and it's incredibly thin –so it's the kind of thing that is really useful for installation projects, whether that's something as simple as a photo slideshow in a real picture frame or a home automation control interface embedded into a cupboard door.

13 Sound configuration

Before configuring speech support for Navit, configure the external sound card. You have to stop the Broadcom module from loading and remove some Raspberry Pi-specific ALSA (Advanced Linux Sound Architecture). To do this, sudo-edit /etc/modprobe and comment out (i.e. prefix with a #):

```
snd-bcm2835
```

Then run:

```
sudo rm /etc/modprobe.d/alsa*
```

Reboot for the changes to take effect. Use alsamixer to set the volume on the if it's too quiet.

14 Download a voice

The speech synthesis software needs a voice and a proprietary binary. You can get both by completing the following steps:

```
sudo mkdir -p /usr/share/mbrola/voices/
wget http://www.tcts.fpms.ac.be/synthesis/mbrola/dba/en1/en1-980910.zip
unzip en1-980910.zip
sudo cp en1/en1 /usr/share/mbrola/voices
wget http://www.tcts.fpms.ac.be/synthesis/mbrola/bin/raspberri_pi/mbrola.tgz
tar zxvf mbrola.tgz
sudo mv mbrola /usr/local/bin/
```

15 Create speech script

Navit supports speech by running an external script and passing the text to speak as an argument. Create one using:

```
cd /home/pi/.navit
wget  http://liamfraser.co.uk/lud/carpi/chime.wav
touch speech.sh
chmod +x speech.sh
```

Now edit speech.sh:

```
#!/bin/bash
aplay -r 44100 /home/pi/.navit/chime.wav
espeak -vmb-en1 -s 110 -a 150 -p 50 "$1"
```

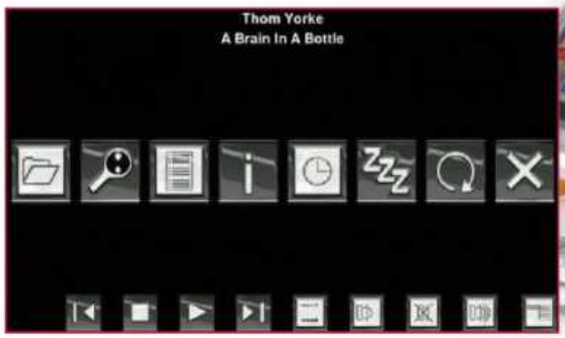
Finally, test it with:

```
./speech.sh "Hello World"
```

Full code listing

```
        try:
            r = requests.get(api_req)
        except requests.exceptions.RequestException as e:
            return errstr

        return WeatherClient.usefuldata(r.json())

class CarLauncher:
    def __init__(self):
        pygame.init()
        pygame.mixer.quit() # Don't need sound
        screen_info = pygame.display.Info()
        self.screen = pygame.display.set_mode((screen_info.current_w,
                                               screen_info.current_h))
        pygame.display.set_caption('Car Launcher')
        self.titlefont = pygame.font.Font(None, 100)
        self.wfont = pygame.font.Font(None, 30)
        self.w_text = None # Weather text

    def clean_background(self):
        background = pygame.Surface(self.screen.get_size())
        self.background = background.convert()
        self.background.fill((0, 0, 0))

        # Render title centered
        text = self.titlefont.render("CarPi Launcher", 1, (255, 255, 255))
        textpos = text.get_rect()
        textpos.centerx = self.background.get_rect().centerx
        self.background.blit(text, textpos)

        self.screen.blit(self.background, (0,0))
        pygame.display.flip()

    def main_menu(self):
        # btns maps Text -> Rectangles we can do collision detection on
        self.btns = {'Music' : None, 'NAV' : None, 'Weather' : None}

        item_num = 1
        for key in self.btns:
            text = self.titlefont.render(key, 1, (255,255,255))
            textpos = text.get_rect()
            max_width = self.background.get_rect().width / len(self.btns)
            center_offset = max_width * 0.5
            # This y pos puts buttons just below title
            textpos.centery = self.background.get_rect().centery / 2
            textpos.centerx = (max_width * item_num) - center_offset
            self.btns[key] = textpos
            self.screen.blit(text, textpos)
            item_num += 1

        pygame.display.flip()

    def select_rect(self, rect, text):
        # Colour a rect the user has clicked in green
        surface = pygame.Surface((rect.w, rect.h))
        surface.fill((0, 255, 0))
        # Now we have to draw the text over it again
        t = self.titlefont.render(text, 1, (255,255,255))
        surface.blit(t, (0,0))
        self.screen.blit(surface, rect)
        pygame.display.flip()
```

Above The Navit software comes with a host of options built into its menu hierarchy



Above The pympdtouchgui front-end for the music player is surprisingly featureful

You will need to write your own launcher for CarPi

Make it mobile

It is definitely best to put this project together in a clean workspace so that you can clearly see what you're working with and ensure everything is correctly wired and soldered, but the point of the project is to make this setup portable so that you can put it in your car and use it on the road. You could install everything into a single, hand-made enclosure or customise a large bought one, or you could secure the various parts inside, for example, your glovebox or car doors. You'll also need to power both the screen and your Pi with a power pack and ensure that the GPS antenna is fastened into a good spot for signal.

16 Configure Navit for speech

The last part is simple. Edit the Navit config file again (/etc/navit/navit.xml) and replace the following line:

```
<speech type="cmdline" data="echo 'Fix the speech tag in navit.xml to let navit say:' '%s'" cps="15"/>
```

...with:

```
<speech type="cmdline" data="/home/pi/.navit/speech.sh %s" cps="10" />
```

Now you can run Navit with **DISPLAY=:0.0 navit** and have fun experimenting.

17 Install the music player

MPD is the music player back-end and pympdtouchgui is the front-end that needs installing manually:

```
sudo apt-get install mpd ncmpcpp
wget http://www.spida.net/projects/software/pympdtouchgui/pympdtouchgui-0.320.tgz
tar zxvf pympdtouchgui-0.320.tgz
cd pympdtouchgui-0.320/
sudo python setup.py install
# Fix hard coded path in software
sudo ln -s /usr/local/share/pympdtouchgui/ /usr/share/pympdtouchgui
```

18 Copy music

Scp (secure copy protocol) was used here to copy music. First get the Pi's IP address by running **ip addr**. Then run **sudo passwd** to set a password for root. From a computer with music on, run:

```
scp -r music_folder root@pi_ip_address:/var/lib/mpd/music/
```

Then on the Pi, change the ownership of the music that you just copied:

```
sudo chown -R mpd:audio /var/lib/mpd/music
```

19 Update mpd music library

Ncmpcpp is a command line client for mpd. Type **ncmpcpp** and press U to update the library. Press 3 to browse the library and check the music is there, and press Q to quit. Pressing 1 will select the help screen if you want to do more.

20 Install awesome window manager

Now you will need to write your own launcher for CarPi, which will run full-screen. To ensure every application is forced to full-screen, use awesome window manager in full-screen mode.

```
sudo apt-get install awesome
sudo rm /etc/alternatives/x-session-manager
sudo ln -s /usr/bin/awesome /etc/alternatives/x-session-manager
```

When changing the default x-session-manager, awesome will be auto-started at boot instead of LXDE. If you reboot the Pi, awesome should then load up automatically.

21 Install the requirements for your launcher

The launcher is going to use a weather API combined with location data from the GPS receiver to give weather updates when requested. The nicest HTTP API for Python is requests, which you can install by doing the following:

```
sudo apt-get install python-pip
sudo pip install requests
```

22 Write the launcher code

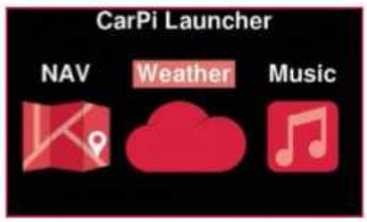
Creating the code itself is pretty self explanatory, but you can use our ready-made version by downloading the CarPi package from **FileSilo.co.uk** and extracting carlauncher/carlauncher.py.

23 Start the launcher automatically

Sudo-edit /etc/xdg/awesome/rc.lua and move awful.layout.suit.max.fullscreen to the top of the layouts list. Add the following to the bottom of the file:

```
awful.util.spawn_with_shell("/home/pi/carlauncher/carlauncher.py")
```

Now reboot again and the launcher should come up automatically.



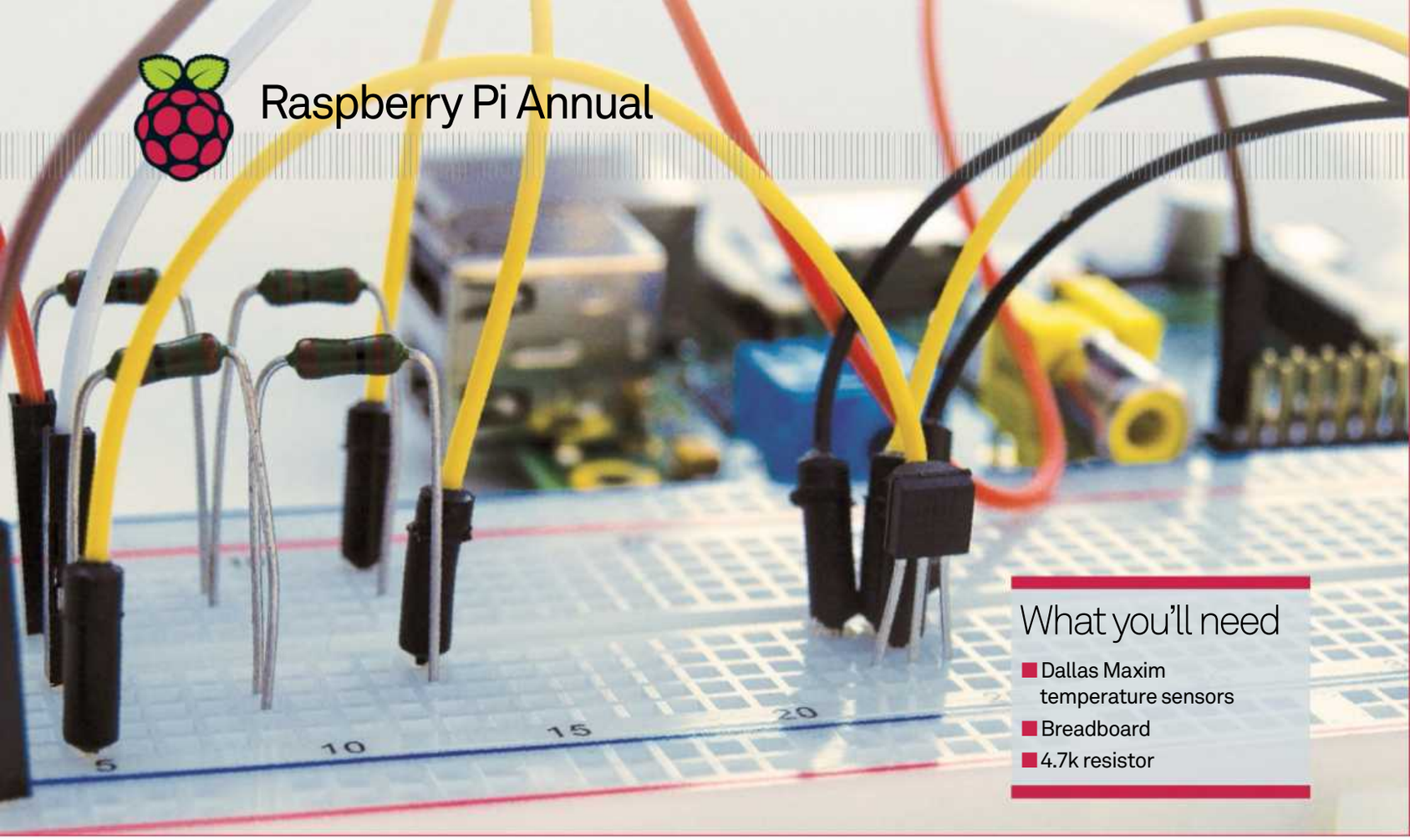
24 Future improvements

There are a number of improvements that could be made to the base project at this point:

- Make the launcher switch between applications rather than start them again each time
- Make the launcher look better aesthetically with icons
- Use Mopidy instead of MPD so you can use Spotify
- Further Navit configuration to make it more featureful
- An SSD or USB flash drive for storage to make things quicker

Full code listing

```
    def reset(self):
        self.clean_background()
        self.main_menu()
        self.render_weather()

    def execute(self, path):
        os.system(path)
        # os.system blocks so by the time we get here application
        # has finished
        self.reset()

    def render_weather(self):
        if self.w_text == None:
            return

        # Get y starting at the bottom of the nav button
        margin = 10
        y = self.btns['NAV'].bottomleft[1] + margin

        for t in self.w_text.split("\n"):
            line = self.wfont.render(t.rstrip(), 1, (255,255,255))
            line_rect = line.get_rect()
            line_rect.centerx = self.background.get_rect().centerx
            line_rect.y = y
            self.screen.blit(line, line_rect)
            y += margin + line_rect.height

        pygame.display.flip()

    def handle_events(self, events):
        for e in events:
            if e.type == QUIT:
                sys.exit()
            elif e.type == MOUSEBUTTONDOWN:
                pos = pygame.mouse.get_pos()
                # Check if it collides with any of the buttons
                for btn_text, rect in self.btns.iteritems():
                    if rect.collidepoint(pos):
                        self.select_rect(rect, btn_text)
                        if btn_text == "NAV":
                            self.execute("/usr/bin/navit")
                        elif btn_text == "Music":
                            self.execute("/usr/local/bin/pympdtouchgui")
                        elif btn_text == "Weather":
                            self.w_text = WeatherClient.update()
                            # Reset will render weather if string is populated
                            self.reset()

    def loop(self):
        clock = pygame.time.Clock()
        self.reset()

        while 1:
            self.handle_events(pygame.event.get())
            # 5 fps is plenty
            clock.tick(5)

if __name__ == "__main__":
    cl = CarLauncher()
    cl.loop()
```



What you'll need

- Dallas Maxim temperature sensors
- Breadboard
- 4.7k resistor

Harness the power of the 1-Wire bus

Custom sensor circuits require experience in signal processing. The 1-Wire bus simplifies accessing ready-made sensors

When interacting with sensors, one of three buses gets used. SPI is used for high-bandwidth applications, while the slower I2C excels at handling a large amount of slower sensors. Dallas Maxim's proprietary 1-Wire bus has even slower data rates and a longer range than I2C, making it ideal for communicating with tiny devices like weather sensors and thermometers, and it is quite interesting because the 1-Wire bus requires only two wires to be run to the sensor – power is supplied via the data line.

The bus has firmly established itself in two areas of application. First is iButtons, which contain memory and/or a cryptographic ID that can be used for access control – they have been used as keys for residential areas and school campuses, for example, as well as 'smart tickets' for public transportation. In terms of the second main application, Dallas Maxim also peddles a series of inexpensive temperature sensors, which we will take a look at here.

We're going to introduce you to the DS18B20 family. They simplify the gathering of accurate temperature data – bid farewell to AD converters, linearisation and similar hassles.

01 Update your Pi

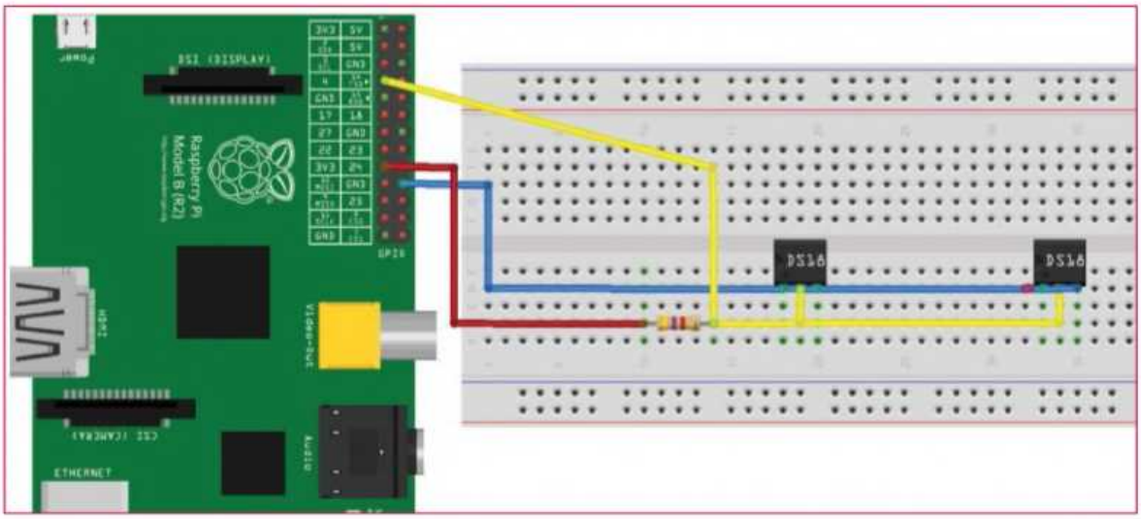
Dallas Maxim's 1-Wire bus is implemented via a kernel module, which saw significant changes with the introduction of Kernel 3.18.3. Due to this, an update is recommended. Run the commands **sudo apt-get update** and **sudo apt-get upgrade**, and check the current kernel version by entering **uname -r**.

02 Connect the sensors

To measure data, you must connect the sensors to the process computer. Parasitic mode networks are wired up in accordance to the diagram at the top of the next page. The resistor shown is a pull-up resistor, responsible for making the bus 'float' to 3V3 when it is not pulled down actively.

03 Apply some power

1-Wire is innovative due to its power parasitisation: each sensor has a charged capacitor to provide the energy necessary for measuring. The drive capability of the pull-up resistor is limited, so with larger networks, supply power directly. The kernel driver cannot use the accelerated conversion speed.



Left We're using a 4.7 kΩ pull-up resistor in this circuit

04 Configure the kernel

The Raspberry Pi realises a basic form of 1-Wire access via a kernel module. This must be enabled at boot time, a process best accomplished by editing /boot/config.txt, as shown below, and then performing a reboot of the process computer. Further guidance on the Pi's config.txt file is available at bit.ly/1K73PqR.

```
dtoverlay=w1-gpio-pullup,pullup=on
dtdebug=on
```

05 Power up

With that in mind, it is now time to power up the process computer. Carefully touch the temperature sensors in order to check if they heat up unduly, as this is a characteristic symptom of wrong connections in the circuit. If it is not the case then your circuit works. You can then look for the sensors in the /sys/bus/w1/devices tree.

06 Read with C

W1-therm maps the individual devices into the file space. This means that the data contained in them can be read like any other text file. We will deploy a small program that starts out by traversing the filesystem in order to find eligible devices.

```
#include <stdio.h>
#include <dirent.h>
#include <string.h>

void main()
{
  DIR *dir;
  dir=opendir("/sys/bus/w1/devices");
  struct dirent* mydir;
  while((mydir=readdir(dir)))
  {
      if(mydir->d_type==DT_LNK && strstr(mydir-
>d_name, "28-"))
    {
      printf(mydir->d_name);
      printf("\n");
    }
  }
  closedir(dir);
}
```

07 Read with C, part 2

In the next step, the inner loop must be expanded. It will iterate over the various devices, reading their data using the normal C APIs. Data is then printed to the command line in a slightly formatted fashion.

```
if(mydir->d_type==DT_LNK &&  strstr(mydir->d_name,
"28-"))
  {
      char myField[1024];
      sprintf(myField, "/sys/bus/w1/devices/%s/
              w1_slave", mydir->d_name);
      FILE *myFile=fopen(myField,"r");
          if(myFile!=NULL)
          {
              fscanf(myFile, "%*x %*x %*x %*x %*x
                %*x %*x %*x %*x : crc=%*x %*s");
              double myTemp;
              fscanf(myFile, "%*x %*x %*x %*x %*x
                %*x %*x %*x %*x t=%lf", &myTemp);
              myTemp/=1000;
              printf("%lf \n",myTemp);
          }
      }
```

08 Compile and run

Developers who are used to working with Python – the scripting language can, of course, also be used to access the file – might find the deployment process a little odd. GCC compiles our code into an executable file, which can then be run like any other application.

```
pi@rpilab ~ $ gcc rpiTherm.c
pi@rpilab ~ $ ./a.out
27.375000
27.500000
```

09 Go pro!

Accessing professional 1-Wire devices – think memory chips, iButtons and cryptographic coprocessors – is handled via the OWFS library. Unfortunately, this does not support the w1-gpio library. Using it will require the deployment of expansion hardware: the I2C-based DS2482 is the most common external controller.

Light up

Sensorics can be a funny business. One interesting experiment involves taking two DS18B20 sensors that are placed a few centimetres apart. One of them is then forced to convert data once every second, while the other one gallivants at a more leisurely pace. As time passes by, the active sensor will show a higher ambient temperature. This behaviour is caused by the conversion process, which incidentally generates heat.

Print wirelessly with your Raspberry Pi

Breathe new life into an old printer by using your Raspberry Pi as a wireless print server

What you'll need

- Latest Raspbian image
  raspberrypi.org/downloads
- USB printer
- USB wireless card

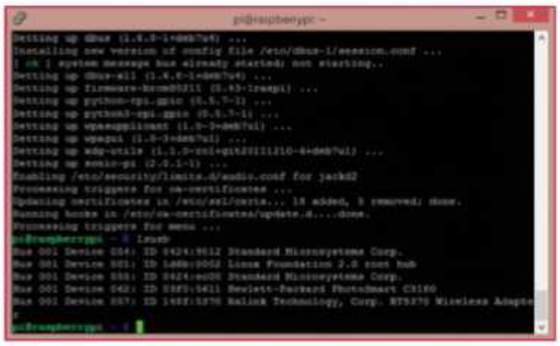
Wireless printing has made it possible to print to devices stored in cupboards, sheds and remote rooms. It has generally shaken up the whole process of printing and enabled output from smartphones, tablets, laptops and desktop computers alike. But you don't have to own a shiny new printer for this to work; old printers without native wireless support don't have to end up in the bin, thanks to the Raspberry Pi.

The setup is simple. With your Raspberry Pi set up with a wireless USB dongle, you connect your printer to a spare USB port on the computer. With Samba and CUPS (Common Unix Printing System) installed on the Raspberry Pi, all that is left to do is connect to the wireless printer from your desktop computer, install the appropriate driver and start printing.

CUPS gives the Raspberry Pi a browser-based admin screen that can be viewed from any device on your network, enabling complete control over your wireless network printer.

01 Check your printer works

Before starting, check that the printer you're planning to use for the project still works and has enough ink. The easiest way to do this is to check the documentation (online if you can't find the manual) and run a test print.



02 Detect your printer

With your Raspberry Pi set up as usual and the printer connected to a spare USB port, enter:

```
lsusb
```

This will confirm that the printer has been detected by your Raspberry Pi. In most cases you should see the manufacturer and model displayed.

03 Install Samba and CUPS

Install Samba to enable file and print sharing across the entire network:

```
sudo apt-get install samba
```

Next, install CUPS:

```
sudo apt-get install cups
```

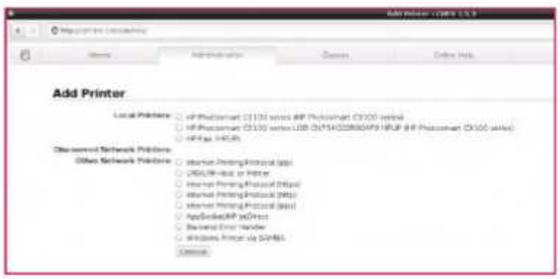
With a print server created, begin configuration by adding default user 'pi' to the printer admin group:

```
sudo usermod -a -G lpadmin pi
```

04 Set up print admin

Set up the CUPS print admin tool first. Boot into the GUI (startx) and launch the browser, entering 127.0.0.1:631.

Here, switch to Administration and ensure the 'Share printers' and 'Allow remote administration' boxes are selected. Next, select Add Printer and enter your Raspbian username and password when prompted.



05 Add your printer

A list of printers will be displayed, so select yours to proceed to the next screen where you can confirm the details, add a name and check the Share This Printer box. Click Continue to load the list of printer drivers and select the appropriate one from the list.

06 Configure Samba for network printing

Using a Windows computer for printing? Samba will need some configuration. Open '/etc/samba/smb.conf' in nano, search (Ctrl+W) for '[printers]' and find 'guest ok' which you should change as follows:

```
guest ok = yes
```

Next, search for "[print$]." Then change the path as follows:

```
path = /usr/share/cups/drivers
```

Begin configuration by adding the default user 'pi' to the printer admin group

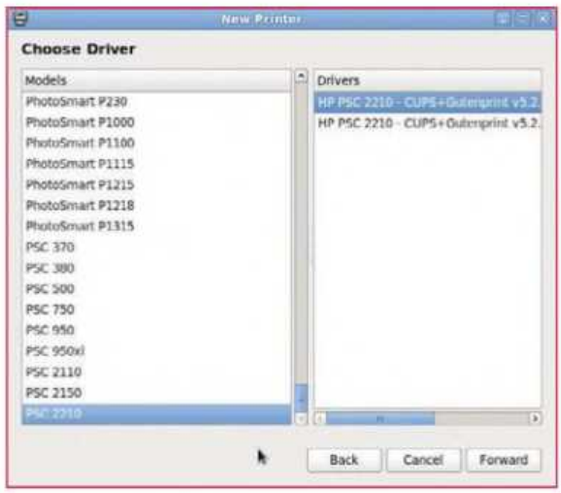
07 Join a Windows workgroup

With these additions made, search for "workgroup" in the configuration file and add your workgroup:

```
workgroup = your_workgroup_name
wins support = yes
```

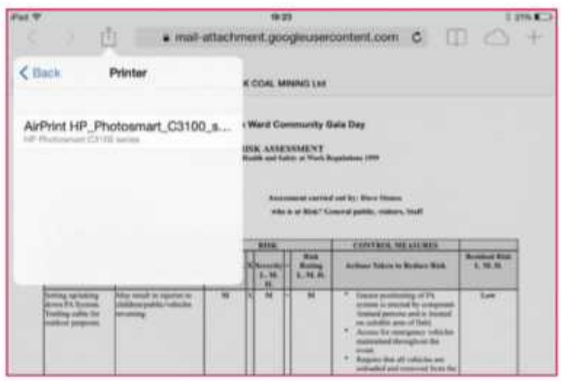
Make sure you uncomment the second setting so that the print server can be seen from Windows. Save your changes and then restart Samba:

```
sudo /etc/init.d/samba restart
```



08 Accessing your printer on Linux

Meanwhile, it's a lot easier to access your wireless printer from a Linux, Mac OS X or other Unix-like system, thanks to CUPS. All you need to do is add a network printer in the usual way and the device will be displayed.



09 Add AirPrint compatibility

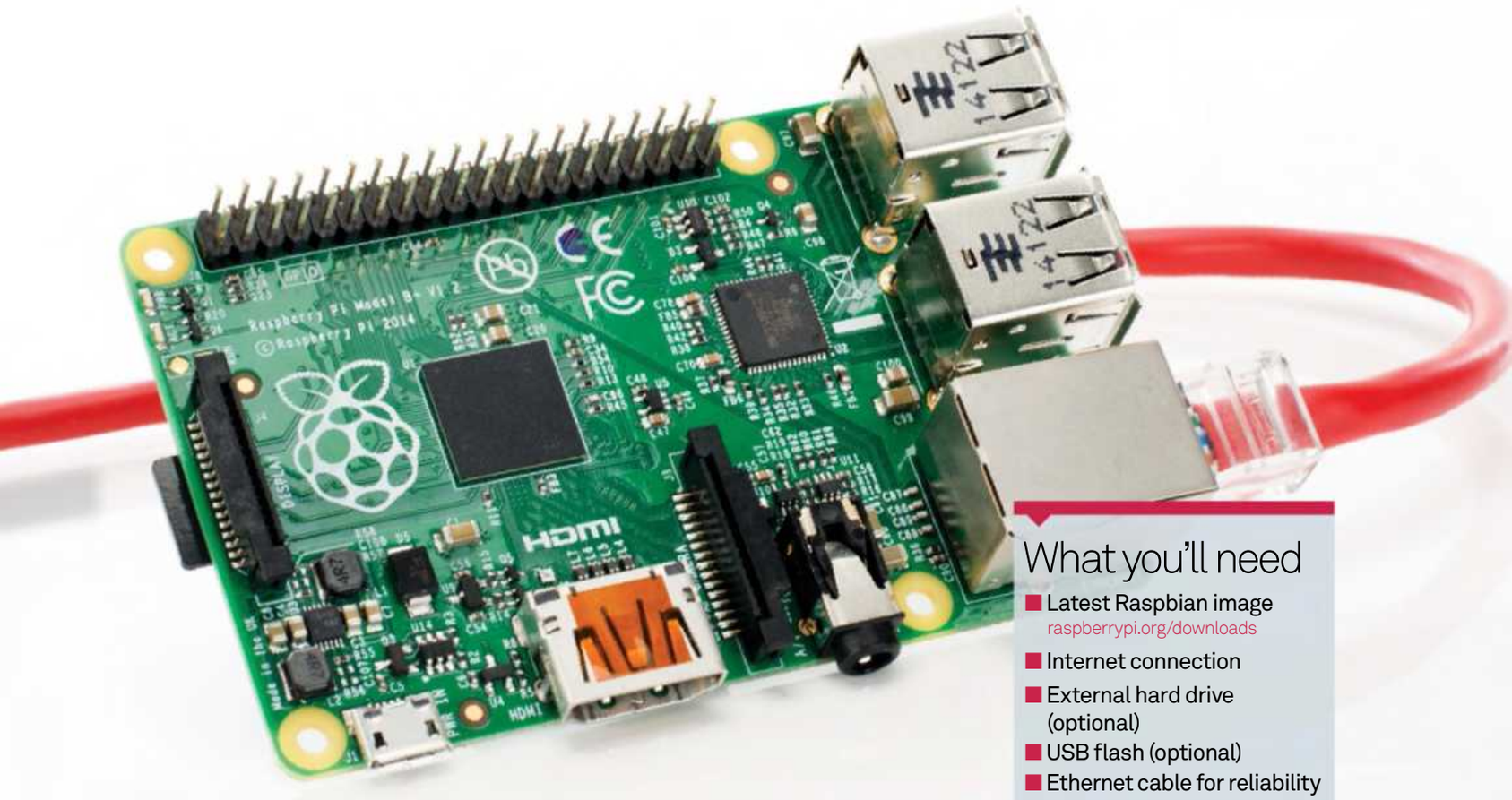
It's also possible to print wirelessly from your iPad using Apple's AirPrint system. To do this, you need to add the Avahi Discover software:

```
sudo apt-get install avahi-discover
```

Your wireless printer will now be discoverable from your iPad or iPhone and will be ready to print.

Host your own website on Raspberry Pi

Don't pay for web hosting. Configure your Raspberry Pi to act as a web server and host modest websites



What you'll need

- Latest Raspbian image raspberrypi.org/downloads
- Internet connection
- External hard drive (optional)
- USB flash (optional)
- Ethernet cable for reliability

Need a lightweight, low-cost web server? Your Raspberry Pi is all you need! Whether you're planning on hosting a static homepage (or one with minimal database use) or need an easy home for development websites, setting up your Raspberry Pi as a web server is surprisingly easy.

Ideal as an always-on device thanks to its low-power requirements, the Raspberry Pi can sit beside your router and serve a basic website to visitors, allowing you to put hosting fees to better use. You might wish to serve pages for some of your Pi projects, or even a personal page to host photos or your CV.

If you're planning on using it as a web-facing device, your Pi will need to be set up with a static IP address. You'll also need to ensure your internet provider offers static IP addresses for their users. Often a price is charged for leasing a static IP, but there are services you can use (such as **noip.com**).

01 Connect your Ethernet cable

For this project it makes more sense to use an Ethernet cable. You may need your existing USB ports to attach flash drives or an external HDD to serve your web page. With Ethernet you will need to rule out any wireless issues that are causing interruptions for your visitors.

02 Get Raspbian updates and Apache

As ever, begin by checking for Raspbian updates:

```
sudo apt-get update
```

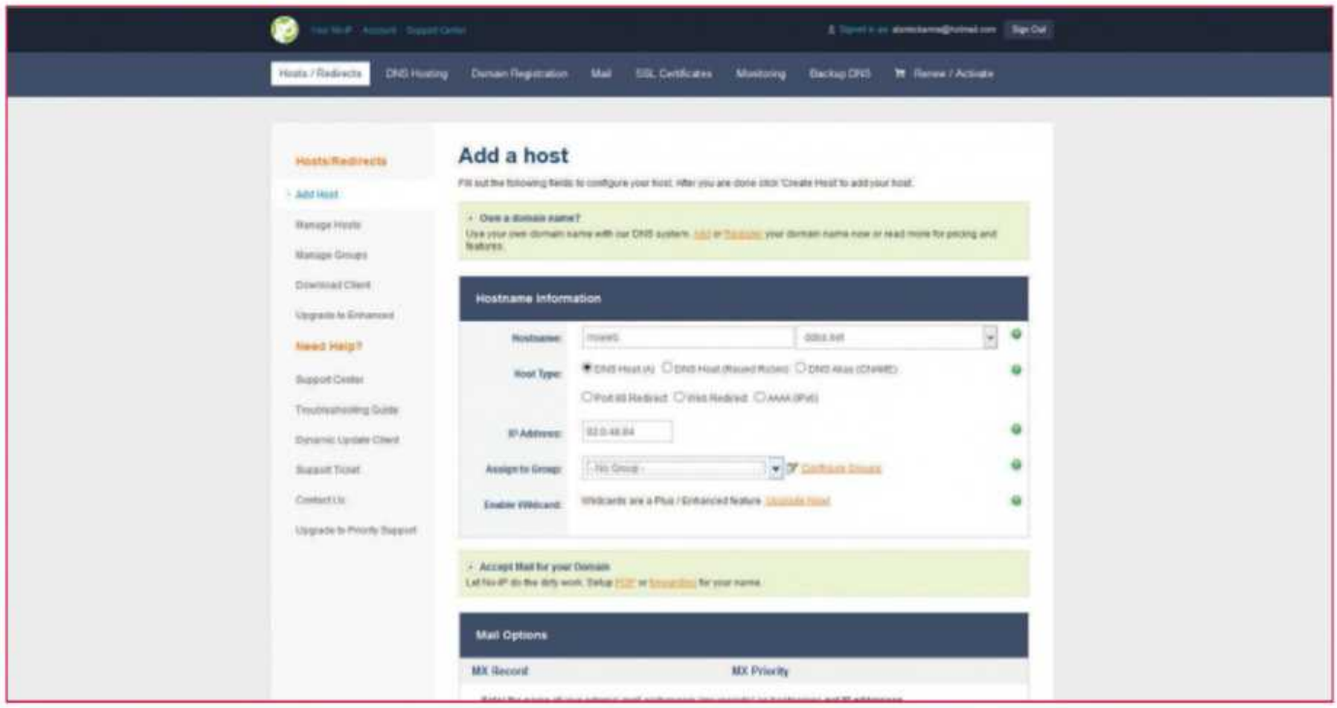
You'll then need to install Apache and PHP:

```
sudo apt-get install apache2 php5 libapache2-mod-php5
```

Finally, restart Apache:

```
sudo service apache2 restart
```

Your Raspberry Pi is now ready to be used as a web server.



You can upload files to /vav/www

03 Check your Pi web server

With Apache installed, open the browser on another computer on your network and enter your Pi's IP address to view the Apache confirmation page.

As things stand right now, all you will be able to view is the Apache index.php page. To add your own HTML and PHP pages, you will need FTP.

04 Install FTP for uploading files

Create a www folder, then install the following vsftpd FTP server software:

```
sudo chown -R pi /var/www
sudo apt-get install vsftpd
```

You'll need to make some changes to Very Secure FTP Daemon, so open it in nano. First, switch:

```
anonymous_enable=YES
```

...to...

```
anonymous_enable=No
```

Next, uncomment the following by removing the # symbols:

```
#local_enable=YES
#write_enable=YES
```

05 Restart the FTP Server

Complete configuration of the FTP software by adding a command to the end of the file which will display server files starting with "." such as .htaccess:

```
force_dot_files=YES
```

Save and exit nano (Ctrl+X) and restart FTP:

```
sudo service vsftpd restart
```

Using the default Raspbian credentials you can upload files to /var/www.

06 Make Pi a LAMP server

By adding MySQL into the mix you can use the Pi to host a database-driven website or even WordPress (although this is best limited to using the device as a development server).

```
sudo apt-get install mysql-server mysql-client
php5-mysql
```

The LAMP bundle is useful of course, but for the best results your site should remain streamlined.

07 Get your site online

Can't afford a static IP for your router? A great solution is available with the free service from **www.noip.com**.

This enables you to point a hostname at your computer by using a client application that will remain in contact with the No-IP servers.

08 Install No-IP

Make a new directory and switch it to:

```
mkdir /home/pi/noip
cd /home/pi/noip
```

Download No-IP on your Pi with:

```
wget http://www.no-ip.com/client/linux/noip-duc-
linux.tar.gz
```

Extract:

```
tar vzxf noip-duc-linux.tar.gz
```

Next, navigate to the directory and use **sudo make** and **sudo make install**, following any instructions. Finish by running:

```
sudo /usr/local/bin/noip2
```

09 Change your password for security

Before using your Pi as a live web server, it's a good idea to change the default password to something more imaginative than 'raspberry'.

In the command line, enter passwd and then follow the prompts to add your new, secure password. You're doing this step because you obviously would not want your Pi web server to get hacked!

Learn to code with FUZE BASIC

FUZE BASIC is a great first language to start learning how to program – here we will take you through the creation of a simple game, from start to finish

BASIC, prolific during the late Seventies and Eighties due to the popularity of the 8-bit BBC Micro, was the language that kickstarted much of the software industry we know today. Many programmers then moved on to more complex and powerful languages like C/+/++/Java etc, games consoles took over the home computer market and BASIC was all but forgotten.

Fast-forward 30 years and it's easy to see why the UK government is desperately trying to get kids coding again – resources are now very thin on the ground and we're outsourcing our programming requirements like there's no tomorrow. There's really never been a better time to become a programmer. You'll find no better introduction than learning to program a game, so we'll start with the classic bat-and-ball genre, but with a twist or two, of course.

To get started you will need to install FUZE BASIC and download the graphics required for the game from **www.fuze.co.uk/lair** and **www.fuze.co.uk/FUZEBIN/spike.zip**.

What you'll need

- FUZE BASIC V3
  fuze.co.uk/lair
- Graphics
  fuze.co.uk/FUZEBIN/spike.zip

Above Here's our sprite-based game, a classic bit of *Breakout*-style fun

Full code FileSilo.co.uk

01 Get started

After downloading and starting up FUZE BASIC, press F2 or type EDIT to get to the FB editor and then type in the following code. Capitalisation for any black text is critical.

```
REM Spikey POP
PROC setup
PROC sprites
CYCLE
PROC intro
level = 1
hhLives = 3
hhScore = 0
CYCLE
PROC newLevel
PROC getready
UNTIL hhLives <= 0 OR levComp CYCLE
PROC displayInfo
PROC hedgeHog
PROC balloon
plotSprite (tramp, MOUSEX, trampY, 0)
UPDATE
REPEAT
IF hhLives <= 0 THEN BREAK
level = level + 1
REPEAT
INK = Red
PROC hideSprites
CLS
text1$ = "GAME OVER!"
printAt (tWidth / 2 - LEN (text1$) / 2, tHeight /
2);  text1$;
UPDATE
WAIT (2)
CLS
REPEAT
END
```

Here we call the routines to set up the main graphics and variables

This CYCLE starts the main game loop

This one is the level loop

This checks if you've finished the level or run out of lives

The section ends the game if you're out of lives

02 Reset a new level

Next we need to enter the variables that we need to reset at the start of each new level:

```
DEF PROC newLevel
bals = maxB
metalCt = level - 1
IF metalCt >= 5 THEN metalCt = 5
PROC newLifeVariables
PROC hideSprites
plotImage (back1, 0, -20)
levComp = 0
PROC setupBals
ENDPROC
```

Variables are reset each time the player starts a new level or a new life

We'll start with the classic bat-and-ball genre, but with a twist or two, of course

03 Reset variables

Now enter the variables to reset every time a life is lost, or at the beginning of a new level.

```
DEF PROC newLifeVariables
hhX = gWidth / 2
hhY = gHeight / 2.8
hhYspd = .1
hhAngle = 0
hXdiff = 0
hXpow = 0
hYpow = .51
hhGrv = 0
hhXdirection = .1
hhYdir = 0
trampX = gWidth / 2
ENDPROC
```

This is all the information needed to calculate the sprite positions

04 Check the hedgehog

This section is where the hedgehog action is at. Here we check the position, the size and if Spikey has hit anything.

```
DEF PROC hedgeHog
hhGrv = (((gHeight - hhY / hYpow) / 80))
hhW = getSpriteW (hhID)
hhH = getSpriteH (hhID)
hCol = spriteCollidePP (hhID, 1)
IF hCol >= b(0, 0) AND hCol <= b(60, 0) THEN
IF  NOT b(hCol, 8) THEN
IF ABS (b(hCol, 2) - hhX) > 20 THEN
hXpow = (b(hCol, 2) - hhX) / 100 * RND (10)
hXpow =  - hXpow
ENDIF
```

PP means 'pixel perfect', and the two variables are the ID of the sprite we're checking and the accuracy from 1 (close) to 16 (boundary box)

05 Check the balloons

Now we check to see which row of balloons has been popped. We also make a small speed adjustment so each time a balloon is hit the speed slowly increases.

```
hhYdir =  NOT hhYdir
b(hCol, 8) = 1
IF hCol >= 0 AND hCol <= 19 THEN
hhYspd = hhYspd + 0.03 + level / 100
hhScore = hhScore + 200
ENDIF
IF hCol >= 20 AND hCol <= 39 THEN
hhYspd = hhYspd + 0.01 + level / 200
hhScore = hhScore + 100
ENDIF
IF hCol >= 40 AND hCol <= 59 THEN
hhYspd = hhYspd + 0.005
hhScore = hhScore + 50
ENDIF
bals = bals - 1
IF bals <= 0 THEN levComp = 1
ENDIF
ENDIF
```

Top row balloons are worth 200 points, the middle row is 100 points and the bottom is 50

The speed increases from balloon hits are dependent on the current level

06 Calculate bounce angle

This section checks to see if we have hit the trampoline and adjusts our bounce accordingly. If Spikey goes below the bottom of the screen, that means he has missed the trampoline and so the lives are reduced. If the screen sides are hit, we reverse the bounce, adjust the sprite angle, then plot the sprite.

```
IF hhYdir = 0 THEN
hhY = hhY - hhGrv
IF hCol = tramp THEN
hhYdir = 1
hXdiff = (hhX - MOUSEX) / 3
IF ABS (hXdiff) > trampW / 6 THEN
hYpow = .51
hhYspd = .1
hXdiff = 0
hXpow = .1
ENDIF
hYpow = hYpow + .05
hXpow = hXpow + hXdiff / 20
hhAngle = hhAngle + hXdiff / 50
setSpriteAngle (hhID, hhAngle)
ENDIF
ENDIF
IF hhYdir = 1 THEN
hhY = hhY + hhGrv
IF hhGrv <= .5 OR hhY >= gHeight THEN hhYdir = 0
ENDIF
IF hhY <= 0 THEN
hhLives = hhLives - 1
IF hhLives > 0 THEN
PROC newLifeVariables
PROC getready
ELSE
gameOver = 1
ENDIF
ENDIF
IF hYpow >= 1.6 THEN hYpow = 1.6
hhAngle = hhAngle + hXdiff / 50
setSpriteAngle (hhID, hhAngle)
IF hhX <= hhW / 2 OR hhX >= gWidth - hhW / 2 THEN
hXpow = - hXpow
hhX = hhX + hXpow
plotSprite (hhID, hhX, hhY, 0)
ENDPROC
```

Here we check if Spikey has made contact with the trampoline

This changes the direction of the bounce

If Spikey misses the trampoline, this reduces the lives and either resets the level or ends the game

This reverses the direction of the bounce if you hit the sides of the screen

> If the screen sides are hit, we reverse the bounce, adjust the sprite angle, then plot the sprite

07 Wait for player to start

Now we enter the code so the player can position the trampoline with a mouse click.

```
DEF PROC getready
CYCLE
getMouse (a, b, mousebutton)
plotSprite (hhID, hhX, hhY, 0)
plotSprite (tramp, MOUSEX, trampY, 0)
UPDATE
REPEAT UNTIL mousebutton
ENDPROC
```

The game won't begin until the player has chosen their trampoline position and clicked the mouse

08 Move the balloons

The balloon procedure sets up an animation loop that increases the animation sequence every four frames. Our balloons will each be stored in an array, which we'll look at in the next step. It then checks to see if a balloon is in pop mode, and if so we rotate it, decrease its size and drop its Y position so that it falls down the screen, shrinking and spinning. If it is not due to be popped, we move each balloon in its direction and then check to see if it goes off the side of the screen, and if so prepare it for display on the opposite side of the screen.

```
DEF PROC balloon
bAnCtr = bAnCtr + 1
IF bAnCtr > 4 THEN
bAnID = bAnID + bAnDir
bAnCtr = 0
ENDIF
IF bAnID >= 6 OR bAnID <= 0 THEN bAnDir = - bAnDir
FOR I = 0 TO 60 CYCLE
IF b(I, 8) = 1 THEN
b(I, 6) = b(I, 6) - .1
b(I, 3) = b(I, 3) - 2
b(I, 5) = b(I, 5) + 10
setSpriteSize (b(I, 0), b(I, 6) * 16)
setSpriteAngle (b(I, 0), b(I, 5))
IF b(I, 6) <= 0 THEN
hideSprite (b(I, 0))
b(I, 1) = 0
ENDIF
ENDIF
IF b(I, 1) THEN
plotSprite (b(I, 0), b(I, 2), b(I, 3), bAnID)
b(I, 2) = b(I, 2) + b(I, 4)
IF b(I, 4) < 0 THEN
IF b(I, 2) <= bMinX THEN b(I, 2) = bMaxX
ENDIF
IF b(I, 4) > 0 THEN
IF b(I, 2) >= bMaxX THEN b(I, 2) = bMinX
ENDIF
ENDIF
REPEAT
ENDPROC
```

This counts the frames

Here we check to see if the balloon is in pop mode and change its size and speed if so

If the balloon is still active, we advance the animation

These are the final checks to see if a balloon needs to wrap around to the opposite edge of the screen

09 Set up the balloons

The initial balloon data is essential to get things running smoothly. Here we set up an array to store all the information needed about each balloon. In this case, we have the sprite ID, X position, Y position, speed, score value, and then active and pop mode states. Compare this section with the code in the previous step to get an idea of how these variables are all labelled.

```
DEF PROC setupBals
bMinX =  - bW * 2
bMaxX = gWidth + bW * 2
bStep = (bMaxX - bMinX) / 20
ctr = 0
FOR n = bMinX TO bMaxX - bW STEP bStep CYCLE
b(ctr, 3) = gHeight - bH
b(ctr + 20, 3) = gHeight - bH * 2.5
b(ctr + 40, 3) = gHeight - bH * 4
b(ctr, 2) = n
b(ctr + 20, 2) = n
b(ctr + 40, 2) = n
b(ctr, 4) = .5
b(ctr + 20, 4) = -.2
b(ctr + 40, 4) = .1
b(ctr, 7) = 200
b(ctr + 20, 7) = 100
b(ctr + 40, 7) = 50
b(ctr, 6) = 10
b(ctr + 20, 6) = 10
b(ctr + 40, 6) = 10
b(ctr, 1) = 1
b(ctr + 20, 1) = 1
b(ctr + 40, 1) = 1
b(ctr, 8) = 0
b(ctr + 20, 8) = 0
b(ctr + 40, 8) = 0
plotSprite (b(ctr, 0), b(ctr, 2), b(ctr, 3), 0)
plotSprite (b(ctr + 20, 0), b(ctr + 20, 2), b(ctr +
20, 3), 0)
plotSprite (b(ctr + 40, 0), b(ctr + 40, 2), b(ctr +
40, 3), 0)
ctr = ctr + 1
UPDATE
REPEAT
ENDPROC
```

A FOR loop is used here to cycle through each balloon inside the array and set its individual variables

The loops works across from the minimum x position to the maximum x, with the ctr + 20 and ctr + 40 lines dealing with the extra two rows of balloons

10 Set up the game itself

The main setup section configures the screen and update settings, loads the background image and defines the main variables, including the balloon arrays and indexes that we have already seen in some of the previous steps. It is worth revisiting this section once you have finished putting Spikey Pop together, because you can test yourself by changing the size of the array and modifying the other code accordingly.

```
DEF PROC setup
setMode (1024, 600)
mouseOff
back1 = loadImage ("back1.png")
DIM b(60, 10)
balloon = 0
bAnID = 0
bAnDir = 1
bAnCtr = 0
ENDPROC
```

DIM is the dimension command used to create the balloon array. It has two variables because it is two-dimensional (for a grid of balloons)

11 Draw the sprites

The order that sprites are created in is the order in which they will be displayed on the screen. Therefore if you want a sprite to always be on top of every other sprite, create it last.

You must set a transparent colour using **setSpriteTrans (pop, 255, 0, 255)**. Setting the transparent colour means you can make sure that sprites don't obscure others when they overlap.

Understanding the setSpriteOrigin command is important. The default is bottom-left, so you need to use offsets if you want to control the sprite from its middle. Far more convenient is to set the origin using: **setSpriteOrigin (pop, getSpriteW (pop) / 2** and then **getSpriteH (pop) / 2)**, as this sets the origin at the absolute centre of the sprite. Other important sprite commands include setSpriteSize, setSpriteAngle and advanceSprite – more information can be found in the Programmer's Reference Guide.

```
DEF PROC sprites
FOR n = 0 TO 60 CYCLE
b(n, 0) = newSprite (7)
FOR nn = 1 TO 7 CYCLE
num$ = STR$ (nn)
IF n >= 0 AND n <= 19 THEN loadSprite ("spe" + num$
+ ".png", b(n, 0), nn - 1)
IF n >= 20 AND n <= 39 THEN loadSprite ("spc" + num$
+ ".png", b(n, 0), nn - 1)
IF n >= 40 AND n <= 59 THEN loadSprite ("spd" + num$
+ ".png", b(n, 0), nn - 1)
REPEAT
REPEAT
bH = getSpriteH (b(0, 0))
bW = getSpriteW (b(0, 0))
tramp = newSprite (1)
loadSprite ("tramps.png", tramp, 0)
trampY = -5
trampH = 20
trampW = getSpriteW (tramp)
trampL = trampW / 2
trampR = gWidth - trampW / 2
hhID = newSprite (1)
loadSprite ("hedge.png", hhID, 0)
setSpriteSize (hhID, 110)
FOR n = 0 TO hhID CYCLE
setSpriteTrans (n, 255, 0, 255)
setSpriteOrigin (n, getSpriteW (n) / 2, getSpriteH
(n) / 2)
REPEAT
setSpriteOrigin (tramp, getSpriteW (tramp) / 2, 0)
ENDPROC
```

These are the sprites for the three rows of balloons

This bit draws the trampoline

Finally we draw Spikey!

12 Clean up

This final routine clears and resets all the sprites, and is needed at the end of each level and at the start of a new life.

```
DEF PROC hideSprites
FOR n = 0 TO hhID CYCLE
setSpriteSize (n, 100)
setSpriteAngle (n, 0)
hideSprite (n)
REPEAT
ENDPROC
```

This only takes one variable, n, because the command simply hides the sprite

FUZE BASIC V3 for Linux and the Raspberry Pi, the Programmer's Reference Guide and a Project Workbook are all freely available to download from **www.fuze.co.uk/lair.** ■

Profiling Python Code

To maximise your Raspberry Pi, you need to use profiling to figure out exactly where the problems are

One of the problems when writing code on a Raspberry Pi is the fact that your computational resources are limited, at least by modern standards. This means that you, as a programmer, need to be more aware of what your code is doing and how it is doing it. Since Python is the language of choice on the Raspberry Pi, we will use that and look at several topics over the next few issues. This month, we will start with the profiling and instrumentation of your code to help you decide exactly which parts need optimisation.

There are two broad categories of profiling: time and memory. Time profiling involves looking at how much time is spent in each section of code. This is most useful in trying to improve the overall algorithm you are using and the specific way that you have implemented it. Memory profiling is used to analyse how efficiently you are using memory and potentially finding areas where you are wasting memory through leaks or simply inefficiencies. There are also two different methods of doing profiling: deterministic and probabilistic. Deterministic profiling tracks the execution of every instruction within your program. The advantage of this method is that you get perfectly correct results. The disadvantage is that it introduces a huge amount of overhead to your program. This means that you cannot do any benchmarking at the same time as you do profiling. The extra overhead also means that it may not be practical to use deterministic profiling on a program that needs to run for a long period of time. The second method of profiling is probabilistic profiling. This method essentially samples your program at some regular interval to see what instruction it is executing. The advantage to this method is that it introduces almost no overhead, and so is ideal for long-running programs. The problem is that you are only collecting statistical data about how often each instruction is being executed. This means that you may actually miss some information. You should end up collecting data on the most heavily-used instructions, on average, so this should not cause any problems in most cases.

The simplest form of time profiling is to simply use the 'time' function. You can record a start time and an end time around a chunk of code to see how long the given section takes to run. While this is easy, it is also relatively coarse and takes up quite a bit of programmer time. The Python standard library itself includes two profiling modules: profile and cProfile. Profile is a pure Python profiling module that offers deterministic time information. The overhead for this module is very high, but because it is pure Python it is easy to extend and add in any extra functionality that you may need. If you don't need that much control, you can use the module cProfile instead. This profiler is actually written in C and imported into Python. This results in quite a lower amount of overhead. In both cases, after importing the relevant module, you can use the function run() to profile a given function. By definition, this means that your code needs to be packaged as a callable function that can be handed in to run(). The other option available is to include a call to cProfile on the command line. It would look like:

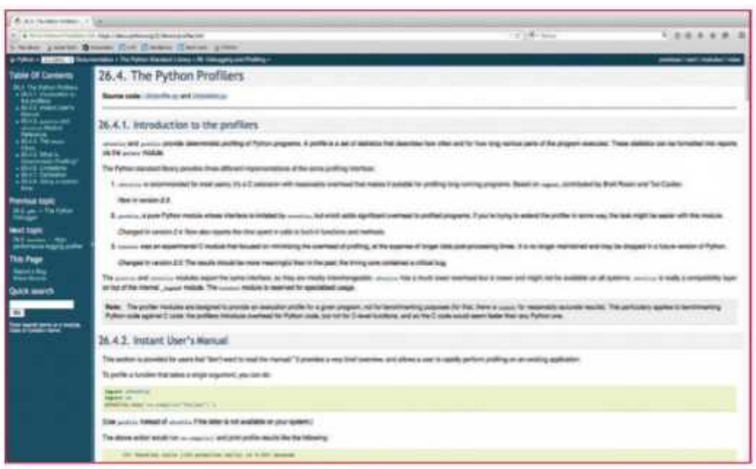
```
python -m cProfile myscript.py
```

In this way, you can profile an entire Python script file rather than just a given function. The default output you get is a summary line giving the total number of functions calls and how long the entire process took. Below this, you are given a breakdown of each function with the following columns:

```
ncalls - number of calls
tottime - total time in each function
percall - tottime divided by ncalls
cumtime - cumulative time spent in each function plus all sub-functions
percall - cumtime divided by number of primitive calls
filename:lineno(function) - location data for each function call
```

Another parameter for the run() function takes a filename in which to save the raw profiling code. You can then use this raw data to do more complex analysis. For example, you can use the stats class from the pstats module. You can then apply some different sorting schemes or do some filtering of the data. You can also print out how many callers or callees each function has.

For memory profiling, the most common option is to use a third-party module called memory_profiler. This



Right Go to section 26 of the Python documentation website (Debugging and Profiling) and select your Python version to learn more

module is available through pip, so you can install it with the command:

```
pip install memory_profiler
```

If you want to see the memory usage for an entire script, you can use it in a similar way to using cProfile:

```
python -m memory_profiler
myscript.py
```

The output from memory_profiler is given by the line, rather than by the function. For each line, it will print out the current memory usage, the increment in memory usage from the previous line and the executable contents of the given line. If you want to narrow down your area of interest to a single function, you can import a function decorator to do memory profiling one function at a time. To use it, you would use:

```
from memory_profiler import profile
@profile
def my_func():
```

It is important to note that the cProfile module also includes a decorator named profile, so you won't be able to use both at the same time. Memory_profiler also includes an executable called mprof that can be used to profile external scripts or codes. To record data, use:

```
mprof run myscript.py
```

You can then get a time plot of memory usage with the command:

```
mprof plot
```

There are several other functions available through mprof.

Now that you know which parts of your code are in need of attention, be sure to check back in next issue. We will look at optimisation tips and tricks that may apply to your specific problem. If not, they may point in directions that you should consider further. ■

Full code listing

```
# The following file can be used whenever you use one of the available decorators
# from memory_profiler or cProfile. You would run it with a command like:
#     python -m memory_profiler myscript.py

myscript.py
-----------
@profile
def my_func(x):
    if x == 0:
       return 1
    elif x == 1:
       return 1
    else:
       return my_func(x-1)

my_func(10)
------------

# The following file can be used to explicitly use the run functions from the
# various profilers:

myscript2.py
-----------
import memory_profiler as mp

def my_func(x):
    if x == 0:
       return 1
    elif x == 1:
       return 1
    else:
       return my_func(x-1)

mp.run("my_func(10)")
------------

# The output would be:
------------
   12 function calls (3 primitive calls) in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall  filename:lineno(function)
        1    0.000    0.000    0.000    0.000  <string>:1(<module>)
     10/1    0.000    0.000    0.000    0.000  myscript2.py:3(my_func)
        1    0.000    0.000    0.000    0.000  {method 'disable' of '_lsprof.
                                                Profiler' objects}
```

Optimising Python Code

After you have profiled your code to figure out what needs work, the next step is to actually optimise the relevant parts

On the previous pages, we looked at a few techniques to profile your Python code and figured out which parts were most in need of attention. When you are running code on a Raspberry Pi you have limited resources, hence the need to optimise and get the most work done that you can. This month, we will look at different techniques that can be used to maximise the amount of RAM and CPU that are available to you. There are several broad categories of optimisation tricks that we will look at on these pages, along with a few examples. Always remember that the first step is to have code that works correctly. Then you should only do the minimum amount of optimisation to get the performance you need. Any extra work is essentially wasted, unless you only want to use the process as a learning opportunity.

The first item to look at is how strings are handled. Strings are immutable lists of characters. This means that when you want to add more characters to a starting string, you need to make a new string and copy both the old and new characters into the new space. In those cases where you are programmatically building up large pieces of text, the moving of characters around in memory can be expensive. Instead of using a construct like:

```
str1 = ""
for word in list:
  str1 += word
```

... you can use:

```
str1 = "".join(list)
```

This builds up the string in a single step, for a reasonable speed-up. Since we are looking at for loops anyway, we should study other things that we can do to make them faster. There is a lot of overhead involved when Python manages a for loop. If the code within it can be bundled as a function, you can use the map command to have this function run for each value stored in a list. In this way, you can remove the overhead of the for loop and move the computational content from Python to C. If you can't do the shift to using a map command, another item to look at is whether there are references to object methods. If so, Python needs to do type checking on every call around the for loop. This means that Python needs to check the types of the parameters being handed in, then see if there is a valid form of the called method, and finally run the called method. You can remove this overhead by assigning these calls to a new name before entering the for loop. This way, all of this type checking and name resolution only happens once outside the loop. So the following code:

```
for word in oldlist:
  newlist.append(word)
```

... can be replaced with this more efficient code:

```
append = newlist.append
for word in oldlist:
  append(word)
```

This type of checking also happens whenever you use any kind of polymorphic operator or function. So, if you wanted to use the + operator, Python needs to check both of the parameters being added together to try and figure out what version of + to use. If you do need to use a polymorphic operation inside a very large loop, it might well be worth your time to look at solutions that use strictly defined data types to remove these checks. While this flexibility in data types is a strength of Python, you may need to give up this flexibility in very specific situations to maximise the speed of your code.

If you are doing numerical computation of any kind, you need to start using the NumPy module. NumPy provides functionality that significantly improves the performance of Python numerical code close to the performance you would expect to see when using C or Fortran. The main way NumPy helps is by providing fixed data types, as we mentioned above. The most basic data structure that NumPy provides is the array. An array is like a list, except that it can only store elements of one type, and it is the basic building block that everything else is built of within NumPy. These arrays are treated as a single data element, so type checks only need to happen once for the entire array. As an example, let's say that you want to multiply the elements of two vectors. In 'regular' Python, you would use a loop like:

```
for index1 in range(50):
  c[index1] = a[index1] * b[index1]
```

... where a and b are vectors of length 50. In this case, every time you run through the loop Python needs to check the types for a[index1] and b[index1]. This is quite a bit of overhead. By using NumPy and the provided array data structures and functions, you can rewrite this code as:

```
c = a * b
```

This makes the code clearer to read. It also only involves a single type check to see whether a and b can be multiplied together. The actual operation then gets passed off to a C library, usually from some linked in linear algebra packages like BLAS or LAPACK. This provides a substantial speed increase because you can take advantage of all of the optimisation work that has gone into these external C or Fortran libraries.

There are some obscure ways of speeding up your code, too. For example, Python needs to regularly check to see whether something else needs to be done. This something else may be looking to see if a different thread needs to run or whether a system call needs to be processed. This can take quite a bit in terms of resources. You can use the command **sys.setcheckinterval()** to change how long Python waits to run these checks. If you aren't using threads or making system calls that may be sending signals to your code, you can set this to a higher value to improve the performance of your code. Another

> A more Pythonic method is to use classes and objects. You can write a script that defines a class that contains method

more obscure technique that optimises both memory and speed is to use the **xrange** command. If you need a list of integers, this is usually handled by the command **range**. The problem with this command is that the entire list gets created in total and stored in memory; you access the list through an iterator and walk through each of the elements. The xrange command provides the list of integers through a generator. A generator does exactly what you think it does; it generates the elements that iterate over as they are needed. This means that you only have one element of the list in memory at any particular time. Also, you don't have the wait while the entire list is being generated. If you are looping over a for loop that has a few thousand cycles, or potentially even millions, this switch to using xrange can provide a significant speed boost. If you are iterating over data in a file, you may actually want to do the opposite. Reading from a hard drive can be much, much slower than reading from memory. This means that you should read in as much of a file as you can reasonably fit into your available RAM in a single operation to speed up manipulating this data. The last obscure method is to consider the scope of variables. Whenever possible, you should use local variables within a function rather than global variables. Python is much more efficient when accessing variables that are within the same scope as the function body, rather than having to move out one or more layers to whichever code body actually called the function in question.

If all of these techniques don't work, then you can always move to a different language and use this alternate code within Python. As an example, you can use Cython to take optimised C code and make it available within your Python program. There are equivalent techniques for other languages used in high performance applications. There are several cases, though, where the algorithm available in the alternate language is just not easily translated to efficient code within Python. In these cases, you have the tools available to leverage this previous work within your own code. We will be looking at this particular technique in greater detail next month. The last technique available is to not use Python at all – at least, not the virtual machine. Python is an interpreted language, which means that every program has the default overhead of having to run through the interpreter rather than running as machine code directly on the hardware. A last resort that is available is to cross-compile your code to machine code for the particular hardware you wish to use. There are a few different projects available, such as Nuitka, that can generate this cross-compiled code. Sometimes, you need to give up portability to get your code running as fast as possible on any particular piece of hardware.

Now that we have looked at different ways of speeding up your code, hopefully this will give you the tools to do great things with your Pi. Not long ago, these techniques were common – it was the only way to get work done. The resources available on the Raspberry Pi are limited compared to modern standards. With more thought, much computational work can be done. Just remember the quote, "premature optimisation is the root of all evil," and be sure you invest your time where it will have the greatest impact.

Compiling to machine code

While you have the ability to use an alternate language within Python – through Cython, for example – you can get optimised code and still stay with pure Python. The Numba project (**numba.pydata.org**) provides a LLVM just-in-time compiler that translates your Python code to optimised machine code. Once you install the Numba module, you can import the jit portion with the following statement:

```
from numba import jit
```

This provides a function decorator, "@jit", that tells Numba to compile the given function to machine code. Every time the compiled function is called, the machine code version is the one that is executed, bypassing the virtual machine. By default, Numba needs to check the types of any input variables and compile appropriate code. This means that you may end up with more than one version of the compiled code. Also, you still have the overhead involved in doing the type checking when the function is called. If you know what data types are going to be handed in, you can skip this check by including type information in the decorator. For example, you could have something like:

```
@jit(int32(int32, int32))
def f(x, y):
  return x + y
```

This compiles the one version you need. Numba also uses lazy compilation, which means that it will only compile when the relevant function is first called. This way, only the portions of code that you actually need go through the overhead of the compilation step. You can use the included pycc utility to compile your Python code ahead of time and this will save you even more at runtime. There are limitations on which data types are supported, and how complex the code can be, but it is still effective enough to make it well worth your time to give it a try. This follows a workflow that is more reminiscent of a traditional compiled programming language, while letting you stay within Python.

Monitoring the network

See what's happening on your network activity with Python and your Raspberry Pi

Raspberry Pis are great devices to use in monitoring applications. With the full set of IO pins, you can attach a range of devices to it, which leads to different environmental monitoring solutions. But what if you need to monitor something more technological, like network activity? This month, we will use Python to look at what's available to code up your own tools and see what's happening on your network. You do need to remember that you will only have access to what is visible on your Raspberry Pi. This may seem obvious, but if you have it connected to a switch, by default it will only see network packets addressed to it. If you want to monitor everything, you'll need to configure your switch to mirror everything to the relevant port.

We'll start by looking at the lowest level available to Python. The standard Python library includes a module called 'socket'. By importing this module, you can talk to a network interface and observe what's happening on it. Remember that you will need root access to talk to physical devices, like the network interface. This means that you need to either be the root user or to run your Python code under sudo. First, you need to create a new socket object with the function: 'socket.socket(socket.AF_INET', 'socket.SOCK_RAW', 'socket.IPPROTO_TCP)'. The first parameter is the address family, the second is the socket type and the last is the protocol type. There is a full description of the options in the Python documentation. You can then bind this socket object to a specific address that represents the interface you're interested in. If you want to capture certain information, you can use the setsockopt() function of the socket object. This will enable you to capture the network packets addressed to you. However, if you want to capture everything visible, you need to set the interface to promiscuous mode. While you can do this within Python with the fcntl module, it can be messy. It's simpler to use:

```
sudo ifconfig eth0 promisc
```

If you're running your code as root, do this from within your program with:

```
import os
os.system("ifconfig eth0 promisc")
```

You can now start collecting data, using the functions recv() and recvfrom(). They both take a number defining the buffer size. The difference is that recvfrom also includes the address that the data came from.

You can also use these raw socket objects to actively go out on the network. Assuming that ICMP packets aren't being blocked on your network, you can create a socket object with a prototype of IPPROTO_ICMP. Then you can use this to try and ping hosts on your network. The function connect() takes a tuple of a host and a port. If you can hit this machine at this port, the connect will succeed. Otherwise, it will fail with an error. Then send out an ICMP packet to this host. You will need to use the pack() of the Python struct module to create a binary struct that you can send on the socket connection – don't forget to close this socket before moving on. If you have specific services that you want to check, you can create a stream socket. Then connect this new socket to the host and port where the service is located and you can use the sendall() function to send some query. Use the recv() function to check the service's response and check services like email or web servers directly.

> To capture everything visible, set the interface to Promiscuous mode

You may instead be interested in looking at the network activity happening on your Raspberry Pi. If so, look at the system information. If you don't want to work directly with socket objects, there are several modules that wrap this and give you higher level functions to look at various system metrics. One example is the psutil module. You can get a list of all the network connections on your Raspberry Pi with the function psutil.net_connections(). You can then count how many there are, or iterate over the list and query the elements for details like the remote address, the prototype or the status of the network connection. If you are more interested in throughput numbers, you can get this data through psutil too. For this, the function in question is net_io_counters(). With no parameter, you will get the aggregate over all of the network interfaces. If you want it divided out, then you can pass in the parameter pernic=True. The returned object has the values of bytes_sent, bytes_recv, packets_sent and packets_recv. If you asked for the results broken down by network interface, you can get the list of interfaces with the function keys(). You can then pull the results for each interface using the keys.

Now you will hopefully have enough ideas and tools to add some basic network monitoring options to your own code. We have only been able to cover the basics here, so don't be afraid to do some research and experimentation to see what else you can do.

Nagios

This month, we have looked at some examples of how to include network monitoring into your own program code. But, what if your primary intention is to do some serious monitoring and you still want to be able to use Python? A good choice might be nagios. Assuming that you are using Raspbian, you can install nagios with the command:

```
sudo apt-get install nagios3
```

This installs the monitoring tools, along with the web interface that you can use to control it. While nagios includes a large number of monitoring tools, you can use Python to write your own plugins and add more functionality. In order to do so, you will also need to install the NRPE server package with:

```
sudo apt-get install nagios-nrpe-server
```

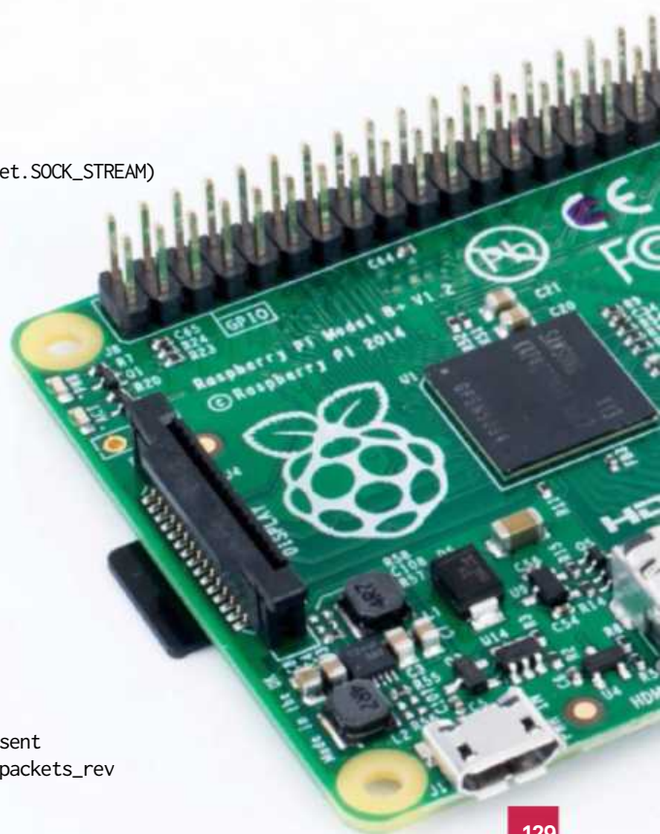
You should put your Python scripts in the same directory (/usr/lib/nagios/plugins) as the system plugins. This simply makes configuration much easier. In order to trigger alerts within nagios, you will need to set an exit code with sys.exit(), where the possible values are:

Exit Code	Status
0	OK
1	WARNING
2	CRITICAL
3	UNKNOWN

Once you have your script written, you need to define it as a command in the file '/etc/nagios/nrpe.cfg' where you give it a command name and point it to the location on the file system. To add this new command to the checks, you will need to add a 'define' section to the file '/stc/nagios/objects/commands.cfg'.

Full code listing

```
# The first step is to import the socket module
import socket

# You need a socket object to work with
# The first parameter is the address family
# The second parameter set the socket type to raw
# The third parameter sets the protocol to TCP
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_TCP)

# The interface needs to be in promiscuous
# mode. If you forgot, you can use the
# following code
import os
os.system("ifconfig eth0 promisc")

# In order to see the outside world, you need
# to bind this socket to a physical interface
s.bind(('192.168.0.11', 0))

# You can start collecting data
# recvfrom include the source address
packet_data = s.recvfrom(65565)

# To check a web server, you can
# send a request to get the index page
s2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s2.connect(('www.google.com', 80))
s2.sendall('GET index.html')

# You can get the response data with
data = s2.recv(10)

# With psutil, you can easily get both
# individual and aggregate network data
import psutil

# The network connections are given by
net_conns = psutil.net_connections()

# The number is
num_conns = len(net_conns)

# Aggregate data is give by
agg_stats = psutil.net_io_counters()

# The individual elements are
print "Bytes sent = ", agg_stats.bytes_sent
print "Packets received = ", agg_stats.packets_rev
```



Manage your Raspberry Pi cluster with IPython

Learn how to configure a cluster of Raspberry Pis to handle parallel programming

In previous issues of Linux User & Developer, we looked at using a cluster of Raspberry Pis and doing parallel computations with MPI. But with Python, there is more than one option available. We will instead look at IPython and see what kind of parallel work you can do with it. Many Python programmers should already know about IPython and how it can be used as an advanced terminal to do interactive Python coding, but it is so much more than that. IPython is built on a client-server type of model – that means that it can be very flexible and can do powerful parallel programming. IPython supports lots of different parallel methodologies, including single program multiple data (SIMD), multiple program multiple data (MIMD), task farming, data parallelism and several other paradigms. With the flexibility afforded by the underlying structure, you can develop almost any type of parallel program. IPython is broken down into four separate sections: IPython engine, IPython hub, IPython schedulers and a controller client. To use IPython on the Raspberry Pi, you need to install the relevant packages. Assuming that you are using Raspbian, or something similar, you can install the required packages with the command:

```
sudo apt-get install ipython
```

One prerequisite that does not get installed automatically is zmq. You will need to install this manually with:

```
sudo apt-get install python-zmq
```

So, what can you do once you have IPython installed? We should take a quick look at how IPython is structured to get a better feel for how you can use it to do really cool stuff.

The first portion to look at is the engine. An IPython engine is a Python instance that listens on a network interface and executes incoming Python commands. Incoming commands and outgoing results might include Python objects, too. When you start up multiple engines, you essentially have a parallel programming system at your disposal. One thing to be aware of is that each individual engine can only execute a single command at a time and is blocking while any particular command is being run. The solution to this potential problem is the controller. A controller is made up of a hub and a collection of schedulers. This controller accepts connections from both engines and clients and acts as a bridge between the two sections. Users talk to the controller, and the controller communicates to the attached engines. This is done through a Client object that connects to a controller and hence to a cluster of CPUs.

In order to do parallel programming, you need to create a cluster of IPython engines, and then you need to connect to it. To start up a cluster of engines, you need to use the command ipcluster. So, if you wanted to start up two engines on a given Raspberry Pi, you would use the command:

```
ipcluster start -n 2
```

> Each individual engine can only execute a single command at a time

To test that everything is working, you could start the IPython interface and try to connect to these two new engines. The code would look like:

```
from IPython.parallel import Client
c = Client()
```

You can check the number of engines available by querying the ids property of the newly created client object. In this case, you should see the list [0,1]. This might be okay if you are just testing some code out, but the whole point is to chain a number of Pis together as one and use them as slaves.

We will start by assuming that you will be using a laptop to do your programming on. This machine will act as the frontend to your Raspberry Pi cluster. We are also going to assume that you are using some distribution of Linux. You will want to install the IPython and python-zmq packages on your laptop using your distribution's package manager. Once that is done, you will need to create a new IPython profile to store the configuration for your cluster. You can do this with the command:

```
ipython profile create
--parallel --profile=rpi
```

This will create a new directory named 'profile_rpi' in your IPython configuration directory and will vary depending on your distribution. In this directory, you will want to edit the file 'ipcluster_config.py' to set up the details for your cluster. In the sample code here, we are using SSH as the communication method. You can then set the hosts as a list, which is stored in the property 'SSHEngineSetLauncher.engines' of the configuration. The hosts are enumerated

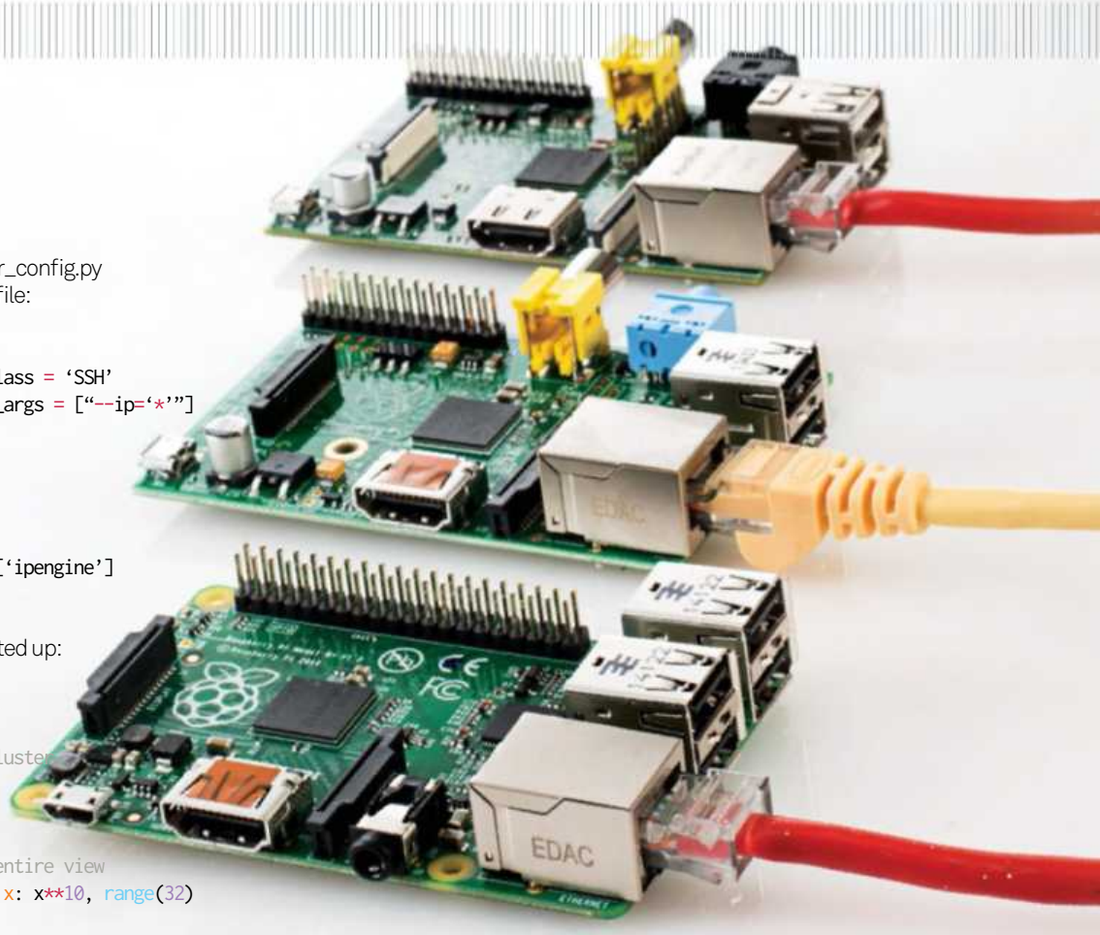
Full code listing

Edit the cluster configuration file ipcluster_config.py
Make sure the following lines exist in this file:

```
c = get_config()
c.IPClusterEngines.engine_launcher_class = 'SSH'
c.LocalControllerLauncher.controller_args = ["--ip='*'"]
c.SSHEngineSetLauncher.engines = {
   'localhost' : 2,
   'rpi1' : 1,
   'rpi2" : 1
}
c.SSHEngineSetLauncher.engine_cmd = ['ipengine']
```

In IPython, work with the engines you started up:

```
from IPython.parallel import Client

# Create a client and view for the cluste
my_client = Client()
my_view = my_client[:]

# You can map a function across the entire view
par_result = my_view.map_sync(lambda x: x**10, range(32)

# You can create a remote function that runs our
# on the engines
@my_view.remote(block=True)
def getpid():
   import os
   return os.getpid()
# Calling 'getpid()' will get the PID from each
# remote engine
```



Above Check out issue 145 of **Linux User & Developer** if you want to learn how to set up, program and use a Ras Pi supercomputer cluster

with the format 'hostname : number_of_ engines'. In the sample code, we used IP addresses, since everything sits on an internal network. The other property you need to set is the command used to start an engine on the individual cluster nodes, which is stored in the property 'SSHEngineSetLauncher.engine_cmd'. If you are using Raspbian, this should simply be 'ipengine'. The last step is to be sure that the profile directories exist on the cluster nodes. On each Pi, you will want to run the command:

```
mkdir -p .ipython/profile_rpi/
security
```

...since it doesn't get created automatically. You also need to be sure that all of the machines on the network can talk to each other cleanly. You may want to look into setting up passwordless SSH, otherwise you will need to enter passwords when you try and start the cluster. These types of communication issues will likely be the primary cause of any issues you may have in getting everything set up.

Now, you have an IPython cluster ready to run. On your local laptop, you can start the cluster up by running the command:

```
ipcluster start --profile=rpi
```

A series of messages will display in the console. Once the controller has finished initialising, you can start using it. You simply create a new client object using the rpi profile and use it to do different types of parallel programming. You should now be able to start using all of those Raspberry Pis that you have been collecting. With IPython, you can rein them all in and get them working together on all of your largest problems. Yow will now have the tools to build one of the lowest-energy supercomputers available in the world.

Pylint

To actually use your IPython cluster, you need to create a view of the engines to manage executing code. The simplest view is created using list syntax and the client object. You can create a view of all available engines with:

```
my_view = my_client[:]
```

You can then use the view's map function to distribute the mapped function across all available engines with

```
par_results = my_view.map_sync(lambda x: x**10, range(32))
```

There are two other basic ways to execute code on the engines, both handled through function decorators. The first way is remote, and it tells each engine to run the same piece of code defined in the function. The second is parallel, which takes an input list and divides it up among the available engines. Then, each engine executes the given function on its share of the initial list. These are both built on top of the view's apply function, which provides a very generic way to execute code on a remote engine. You can control how this code gets executed through flags in the view, or you can set 'my_view.block' to true if you want to wait for the results or to false to return right away. An AsyncResult object tracks the status of the function call.

Optimise by going outside

This article on Python optimisation looks at how to use external compiled code to speed your program up

Two earlier Python articles looked at profiling code, to see where to apply your skills, and then optimise the already existing Python code. In those two cases, you have the option of outsourcing troublesome parts to an external language and compiling that code to machine language. The most common technique to do this is to use Cython (**www.cython.org**) to take C code, compile it to machine code, and then use it within your Python code. The first item that you need is a C compiler that Cython can use. On Linux systems, the default compiler is GCC. To be sure that you have all of the necessary tools, you can use the following command:

```
sudo apt-get install build-essential
```

The second item you need is the Cython package for Python. If you want the latest version available in Pypi, you can use:

```
sudo pip install cython
```

... to install the Cython package into the system Python library location.

Before we dive into how to use Cython, we will confirm everything is working correctly. Your Cython code needs to be in its own file, with a .pyx file name ending. A Hello World function (in the file hello.pyx) would look like the following:

```
def hello_world(name):
    print("Hello World to %s" % name)
```

You then need a setup.py file to define the compilation step:

```
from distutils.core import setup
from Cython.Build import cythonize
setup(name='Hello world app', ext_modules=cythonize("hello.pyx"),)
```

With these two files, you can build the external code with the command **"python setup.py build_ext --inplace"**. This creates a C source code file and compiles it into a shared binary file with a .so file name ending. You can then use this new external code with the Python command:

```
from hello import hello_world
```

... and use it as any other Python function. While the above will let you take your Python code and move it out to compiled machine code, the real power comes when you use Cython to use C libraries within your Python code. The functions within the standard C library are already defined within Cython. You can import these functions with the **cimport** statement. If you wanted to use the function **atoi()** to convert a string to an integer, you could use:

```
from libc.stdlib cimport atoi
cdef parse_char_to_int(char* s):
    assert s is not NULL, "String is NULL"
    return atoi(s)
```

Cython also includes declarations for the C math library. You can import these from the libc.math package. While these two libraries are handy, the majority of the code you need will reside in other libraries. In these cases, you have to provide declarations of them yourself. As an example, we can look at using the **sine** function from the math library and do the importation manually. The first step is to provide a declaration:

```
cdef extern from math.h:
    double sin(double x)
```

This declaration code resides in a file with the ending .pxd. This file needs to have a different name than any .pyx files. As an example, you might place the above code in a file named csine.pxd. You can the import it within a file named sine.pyx:

```
cimport csine
```

... and use it within your Cython code. If you use an object, you can use the function **__cinit__** to handle memory management issues. When you are done using an instance of this object, use the **__dealloc__** function to clean up steps and memory frees. Also tell Cython which external libraries need to be linked in. You can do this with the extra option:

```
libraries=["libname"]
```

... added to the Extension entry in the setup.py file.

Within Python, you can create new objects without worrying about where they will be stored, and you can discard with equal impunity. But sometimes, when you are including C code, you need control over memory management. Cython includes declarations for the functions **malloc**, **realloc** and **free** from the C standard library. For example, let's say you want space for an array of doubles. You can do this with:

```
cdef double *my_array = <double *>malloc(number * sizeof(double))
```

When you are done with this array, you can clean up with:

```
free(my_array)
```

The problem is that this memory is outside of the regular Python heap and so is unaccounted for. A preferred method is to use the **C-API** functions provided in the package cpython.mem. The equivalent functions are **PyMem_Malloc**, **PyMem_Realloc** and **PyMem_Free**. These have the same usage and interface as the lower level C functions from the standard C library. Of course, once you start down this road, you are responsible for freeing memory and avoiding memory leaks.

Now you've seen how to start adding C code to your Python code, you can start some major optimisation. This is actually how packages like numpy and scipy get their impressive speeds. And now you can apply these same techniques to your own code. However, as always, you need to balance the amount of work it takes to write the code with the amount of work it takes to maintain the code and the amount of speedup you get. Try to avoid the temptation of over-optimisation.

> You can create new objects without worrying about where they will be stored

Full code listing

```
# The contents here should be divided into several
# separate files in order to use the examples

# hello.pyx
# Basic Cython hello world
def hello_world(name):
   print("Hello World to %s" % name)

# setup1.py
# This is the setup Python script to compile the
# Hello World Cython example
from distutils.core import setup
from Cython.Build import cythonize
   setup(name='Hello world app', ext_modules=cythonize("hello.pyx"),)

# c_atoi.pyx
# You can import standard Library functions directly
from libc.stdlib cimport atoi
cdef parse_char_to_int(char* s):
   assert s is not NULL, "String is NULL"
   return atoi(s)

# csine.pxd
# You need to declare external C library functions
# to use them within Python
cdef extern from "math.h":
   double sin(double x)

# sine.pyx
# You can then import and use the C function
cimport csine
csine.sin(45.6)

# setup3.py
# You need to add in which external libraries to link to
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize
ext_modules=[Extension("sine", sources=["sine.pyx"],
libraries=["m"])]
setup(name = "Sine", ext_modules = cythonize(ext_modules))
```

Cython and Pyximport

One major issue with using Cython is that you are dragged back into the development cycle of compiled languages. In order to test new code, you need to write, compile, then run these changes in order to see how they behave. If the code you are writing isn't too complicated, then a different option you can choose is to use pyximport. Pyximport is provided through the Cython package, and enables you to have your code compiled on the fly when you need to use it. In order to use it, you need to import it and run the install function:

```
import pyximport; pyximport.install().
```

Then, when you import your pyx file, it gets silently compiled. However, one issue is that you don't have much control over how this compilation is handled. In most cases, though, the defaults are usually fine. Pyximport behaves like make, in that it only does a recompile when a source file is newer than the associated source file. In simple cases, you usually just have a single PYX file. If you have multiple dependencies, you can delineate them within a file with the ending .pyxdep. Each dependency should be on a separate line within this file.

Coding with Cython

If you're interested in learning more about Cython, it's worth picking up a copy of issue 153 – you can order one from **bit.ly/1M2fWKG**. In this issue, Liam Fraser explains how to write a simple polyphonic synthesiser using Python and Cython in tandem, taking advantage of Cython's improved performance in order to effectively play multiple notes at the same time. The code is listed in its entirety for you to examine and type up yourself, and is also available as a **FileSilo.co.uk** download, so it's a great way to see the power of this compiler in action.

What you'll need

- Bare Conductive paint (pen or tub)
- Male to female jumper wires
- An assortment of LEDs, switches and resistors (optional)

Draw circuits with Bare Conductive paint

Assembling circuits has never been so easy with the joys of conductive paint, enabling you to bring together art and electronics in a whole new way

Playing with electronics and physical computing is a very rewarding task – it's capable of causing huge grins to develop on the faces of people of all ages. For a beginner though, the mess of wires and components can become very confusing quite quickly and things like soldering can be a safety concern when children are involved. Bare Conductive has taken the joy of electronics and made it far safer, easier and more versatile with their conductive paint. You can literally draw wires on paper with a paintbrush, use it for cold-soldering or a conductive adhesive and much, much more. There are not a great deal of boundaries to what you can do – if you can do it with normal paint and normal electronics, then you can do it with Bare Conductive paint (even multi-layer circuits are possible).

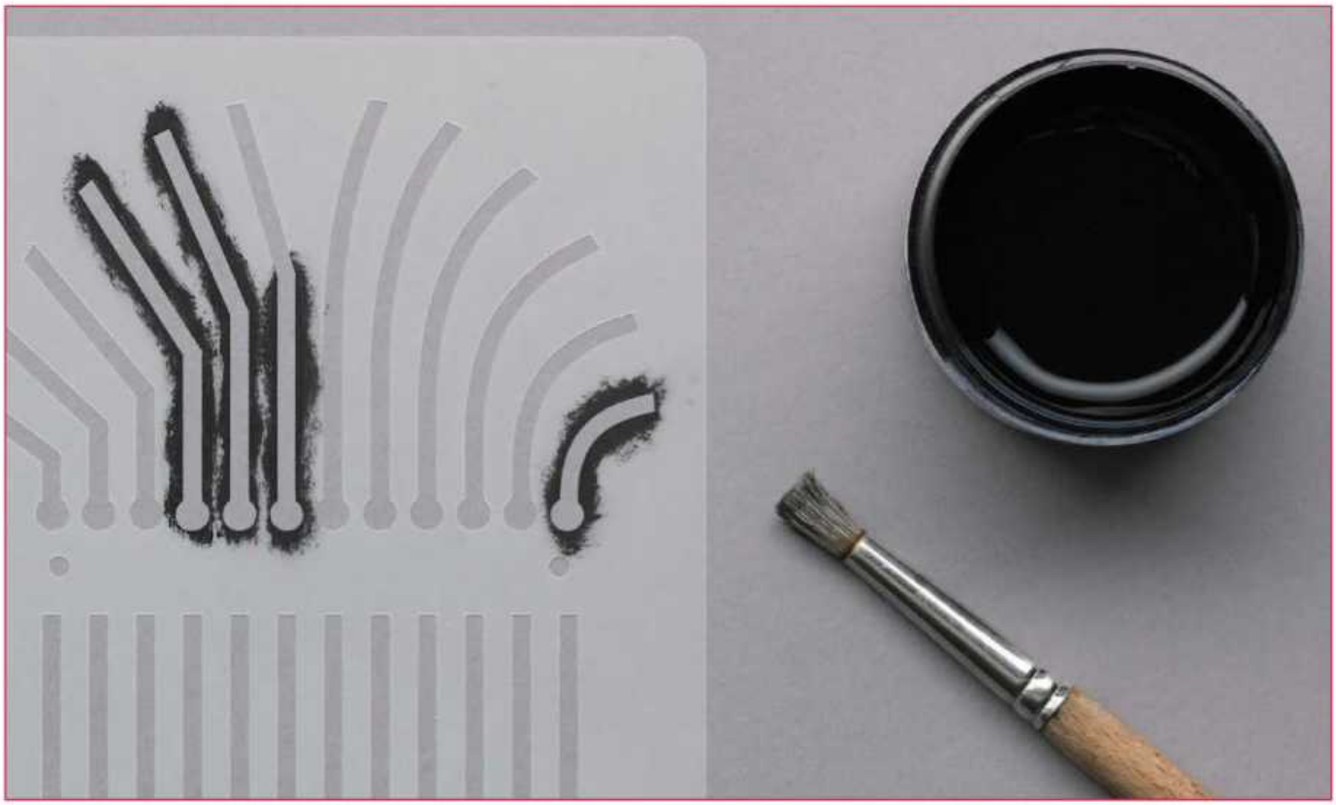
Pair this paint with a microcontroller board and you could be creating interactive art, clothing and projects in no time.

01 Get your tools

Paint and a paintbrush aren't the first items that come to mind when you think about electronics, so you may be wondering where to get them from. Bare Conductive stock the paint and a selection of components in their shop (**bareconductive.com/shop**) but you will need to go somewhere else for art supplies. We would recommend trying a high street craft shop such as Hobbycraft (**hobbycraft.co.uk**) or a local independent.

02 Pick your platform

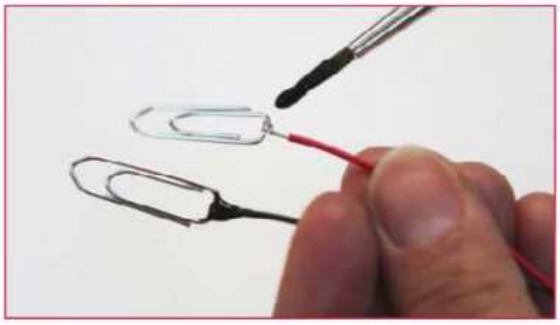
The great thing about Bare Conductive paint is that, when dry, it works just like normal wiring! That means you can use it with any of your favourite microcontrollers like the Bare Conductive Touch Board, a Raspberry Pi or Adafruit's wearable FLORA platform. Or you can just use some small pin batteries and flashing LEDs for a standalone system.



Above Cut custom templates to suit your project's style and requirements

03 Start to paint

You can paint Bare Conductive paint onto pretty much any surface – paper, fabric, walls, clothing, wood, plastic and much more. For really accurate shapes and results, the best idea is to create or purchase a stencil (paper stencils are easiest to make at home but use vinyl for the best edge finish).



04 Connect it up

There are plenty of ways to connect to the conductive paint (from battery packs or microcontrollers for example) no matter what surface it's on, because once it is dry it acts just like an uninsulated wire. Therefore you can use wires glued on with the paint, paper clips, bulldog clips, alligator clips or even sewn-in conductive snaps for wearables projects.

05 Make repairs

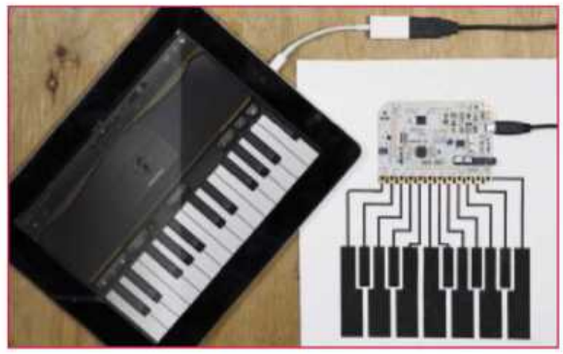
The conductive paint is thick and when it's dry it becomes quite strong. These means you can use it to cold solder things together and repair any breakages. In other words, you could glue components into a circuit board or glue wires together and they would still function electrically. You can even use it to repair damaged tracks on circuit boards.

06 Clean up

A lot of you are probably thinking that something as cool as conductive paint is going to be nasty stuff. Actually Bare Conductive paint is non-toxic, water-based and water-soluble, and can therefore be cleaned easily with soap and water.

07 Make it waterproof

This paint only comes in black and is not waterproof. However, the great thing is that you can use it underneath or alongside any regular paints, varnishes and waterproofing sprays in order to act as insulation – or just to add some colour into your designs!

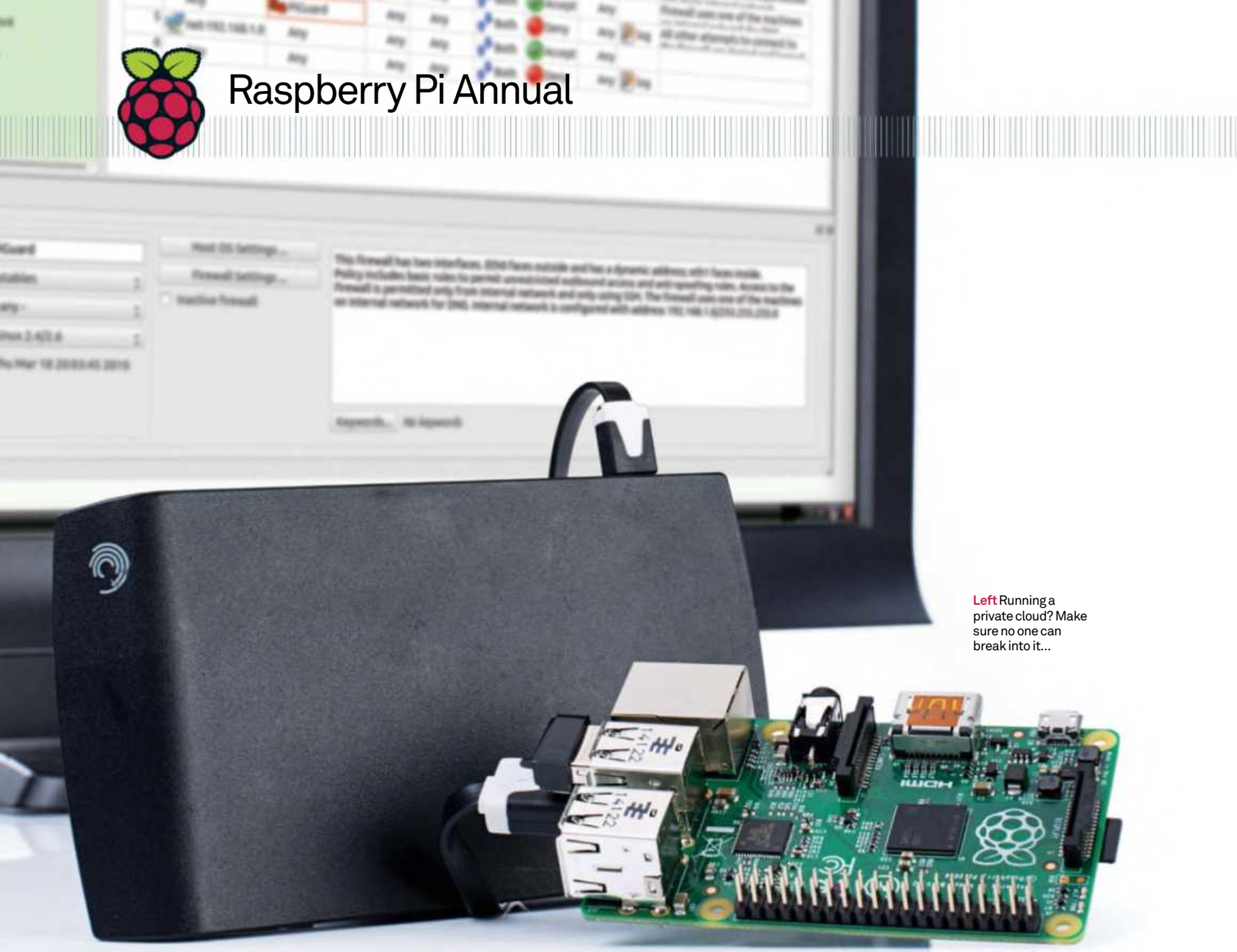


08 Touch and sound

Bare Conductive paint can also be used as a capacitive surface, meaning you can use it for touch, gesture or proximity controls when it is paired with a suitable control board. Bare Conductive make their own called the Touch Board which has everything you need to start experimenting with touch and sound. It can even act as a MIDI controller, an interface or an instrument!

Touch Board

Bare Conductive's Touch Board is an Arduino device compatible with any existing shields and code you might have. It works with any conductive material as well as the Conductive Paint – you can even wire it up to a metal ruler. Using the Conductive Paint, you can also create touchless sensors, for example drawing and programming an electric drum kit that responds to waves and passes over your custom shapes.



Left Running a private cloud? Make sure no one can break into it...

Secure your Raspberry Pi

Concerned about the security of data stored on your Raspberry Pi? Protect yourself with passwords, firewalls and some physical security

There is a distinct security risk around your Raspberry Pi. Storing anything from passwords to firewalls, this important saved data can be stolen or pocketed with minimal effort if someone knows how.

Therefore it's a relief to learn that several tools, tricks and methods can be applied to keep your device and data away from prying eyes. You might, for example, be running a home security cam with images uploaded to a cloud account. These images would be visible to anyone who possesses your Raspberry Pi's login details if you haven't bothered to change the defaults. Such a project (and many others) also demands that a firewall is installed for further improved security on a network.

Whether you're simply changing passwords, keeping your Pi under lock and key or installing a firewall, you'll be surprised at how easy it is to secure your Raspberry Pi and protect all of your important information and files.

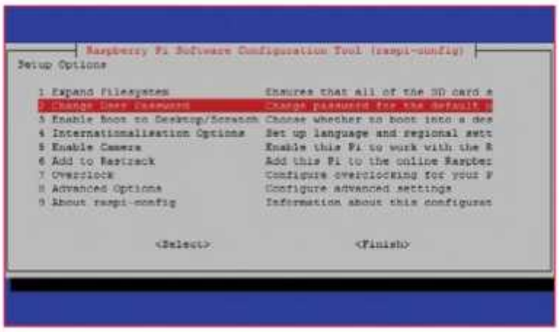
What you'll need

- Velcro
- Adhesive putty
- Lockable cupboard, strongbox, etc.

01 Stop using the default password

Everyone who uses a Raspberry Pi knows that the default Raspbian credentials are 'pi' and 'raspberry'. Naturally, this means that anyone can sign into your computer if you haven't changed these defaults – something you'll need to do as a matter of urgency. After signing in, open the terminal and set a new password with:

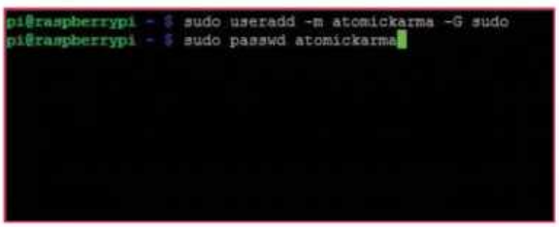
```
passwd
```



02 Change password with raspi-config

If you're setting up a new installation of Raspbian, changing the password is one of the first things that you should do. With a new install, the first boot will automatically run the raspi-config screen.

Here, use the arrow keys to find the second option, change User Password and then follow the on-screen prompts to set yourself a new passcode.



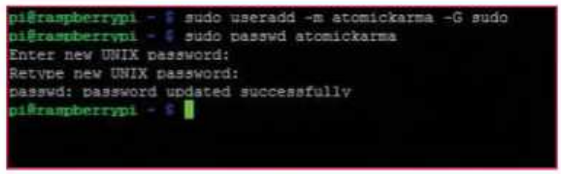
03 Create a new user account

To completely baffle anyone attempting to gain access using default credentials, take the most secure option and create a new user account. In the command line, enter:

```
sudo useradd -m username -G sudo
```

The –m switch creates a new home directory, while the second sudo adds the new account to the superuser group.

With a desktop computer and SD card reader, there is a way that you can recover your password

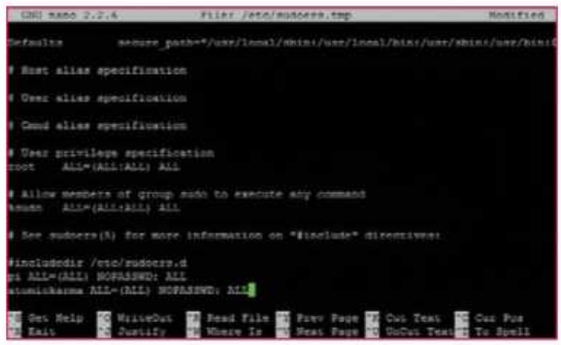


04 Give the new account a password

With the new account set up, the next step is to set a password. As you're not signed into the account at this stage, you won't be using the **passwd** command. Instead, enter:

```
sudo passwd username
```

With the new account ready to use, you should be ready to remove the default pi account from Raspbian altogether.



05 Delete the default Raspbian account

You no longer need the default user account, pi. Sign out and login to your new account, and confirm it is correctly set up by opening:

```
sudo visudo
```

...and adding...

```
username ALL=(ALL) NOPASSWD: ALL
```

...to the final line. Save and exit with Ctrl+X. Now that's done, simply delete the old account with:

```
sudo deluser pi
```

Then remove the home directory:

```
sudo deluser -remove-home pi
```

06 Recover a lost password

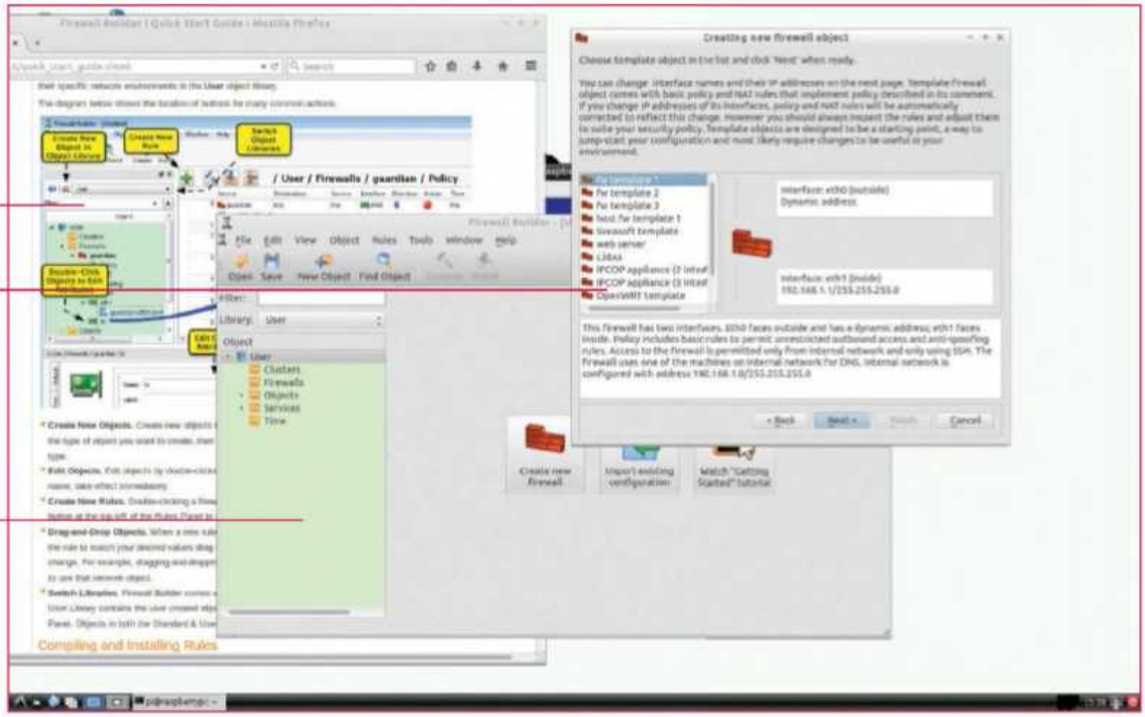
If you've somehow forgotten your Raspberry Pi user account password or suspect that someone has changed it, what can you do?

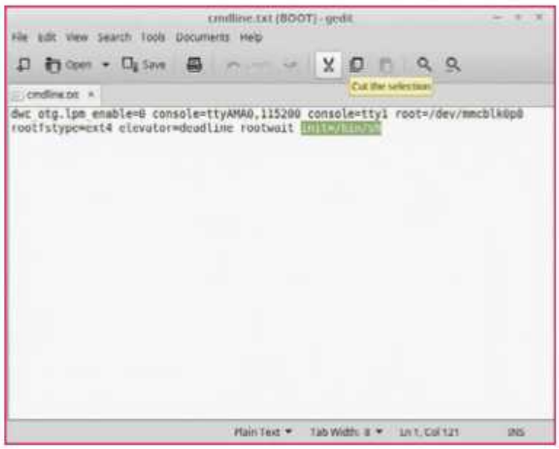
With a desktop computer and SD card reader, there is a way that you can recover your password. Begin by inserting the Pi's SD card into your PC's card reader.

Use a proximity sensor

If you're genuinely concerned about your Raspberry Pi's physical security, you may consider employing some additional hardware to make it less of a target.

Your best option here is probably a proximity sensor configured to detect an unauthorised presence. When coupled with a buzzer, this can detect the presence of an intruder and alert you. It's even possible to configure such an alert as an email message if you're likely to be elsewhere, and it makes for a great side project.





Hiding hardware

Putting your hardware out of sight and/or out of reach is a good option for security, and for something as small as the Raspberry Pi and an SD card you have quite a few options.

For instance, using Velcro or some adhesive putty you might attach the computer to the back of a cupboard or unit, kitchen kickboards or even under a car seat. The SD card, meanwhile, is so compact that you could easily place it under a carpet or even make a home for it in a cushion or shelf – just don't forget where you put it!

07 Edit cmdline.txt

Find the file cmdline.txt and open it in your Linux desktop text editor. Add the following to the end of the last line of the file:

```
init=/bin/sh
```

As the Raspberry Pi boots, this command will be read, enabling us to access a screen to reset the password. Save and eject the card.

08 Change the lost password

Unfortunately you won't be able to use SSH to recover the password, so instead connect a monitor and keyboard to your Raspberry Pi. Boot the Pi and wait for the prompt, at which point you should enter:

```
passwd username
```

Type the password, hit Enter and type it again to confirm.

09 Initialise the Raspbian boot

Thanks to the added code, we have changed the standard Raspbian boot to display a new prompt that will let us change the password.

When this is done, enter the following command to put things back in order:

```
sync
exec /sbin/init
```

The Pi will now boot Raspbian normally, enabling you to sign in with the new password.

10 Revert cmdline.txt

We are not done yet though. Safely shutdown your Raspberry Pi with:

```
sudo shutdown -h now
```

With the Pi powered down, remove the SD card and insert it into the card reader again. Open cmdline.txt in your text editor once again and remove init=/bin/sh, then save and exit. This stops anyone else from resetting your password.

11 Physically secure your Raspberry Pi

Keeping digital intruders out of your Raspberry Pi with firewalls and secure account passwords is only part of the story. To fully protect your Pi you need to think outside of the box.

Barely larger than a credit card, the Raspberry Pi computer can easily be picked up and palmed. Physical security is paramount, but a genuinely secure Raspberry Pi case – for example, one compatible with Kensington locks – has yet to be released. However the ProtoArmour aluminium case from **www.mobileappsystems.com** can be screwed to a secure surface, which is great for more permanent project setups.

Below If you tend to access your Pi remotely via Wi-Fi, consider keeping it locked away

12 Lock it in a drawer

Probably the best way to keep your Raspberry Pi secure is to make sure you keep it locked in a drawer or cabinet – particularly useful if you use the device as part of a security cam system or as a cloud server storing valuable documents.

If no lockable storage is available and you're taking some time away from home where it isn't practical to take the Pi with you, another solution is needed. This might be to travel with your Pi's SD card in your wallet, perhaps leaving the computer attached to the back of a wardrobe with Velcro.

13 Add a firewall

Regardless of which operating system you're using, adding a firewall is a guaranteed way to improve your computer's security. While the Raspberry Pi has a built-in firewall, it is tricky to configure.

Thankfully, some other people have noticed this too and released fwbuilder, an interface to the otherwise complex iptables firewall that comes with Raspbian.

14 Install fwbuilder in Raspbian

Because iptables is a bit fiddly and errors can leave you with no network connection, fwbuilder has been developed to make firewall configuration quick and painless.

We'll use the **apt-get** command to first check for updates and then install fwbuilder:

```
sudo apt-get update
sudo apt-get install fwbuilder
```

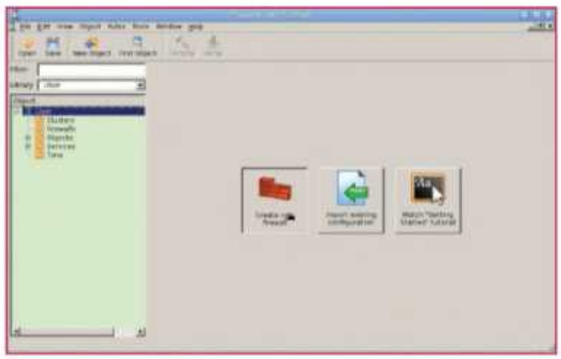
Follow the prompts to install and, once complete, switch to the Raspberry Pi GUI by entering:

```
startx
```

In the Pi's mouse-driven desktop, launch fwbuilder from the Internet menu. Upon launching fwbuilder, follow the given steps to set up your Raspeberry Pi firewall and save the resulting script.

We're nearly done but some adjustments are still required before your Pi fully connects to the network.

A firewall is guaranteed to improve your security



15 Complete firewall configuration

Launch the /etc/network/interfaces script in your text editor and complete configuration by adding

```
pre-up /home/pi/fwbuilder/firewall.fw
```

Next, find the section labelled "Epilog" and add

```
route add default gw [YOUR.ROUTER.IP.HERE] eth0
```

If you're using a wireless card, add the same line but switch the last characters to wlan0:

```
route add default gw [YOUR.ROUTER.IP.HERE] wlan0
```

16 Consider Raspberry Pi theft

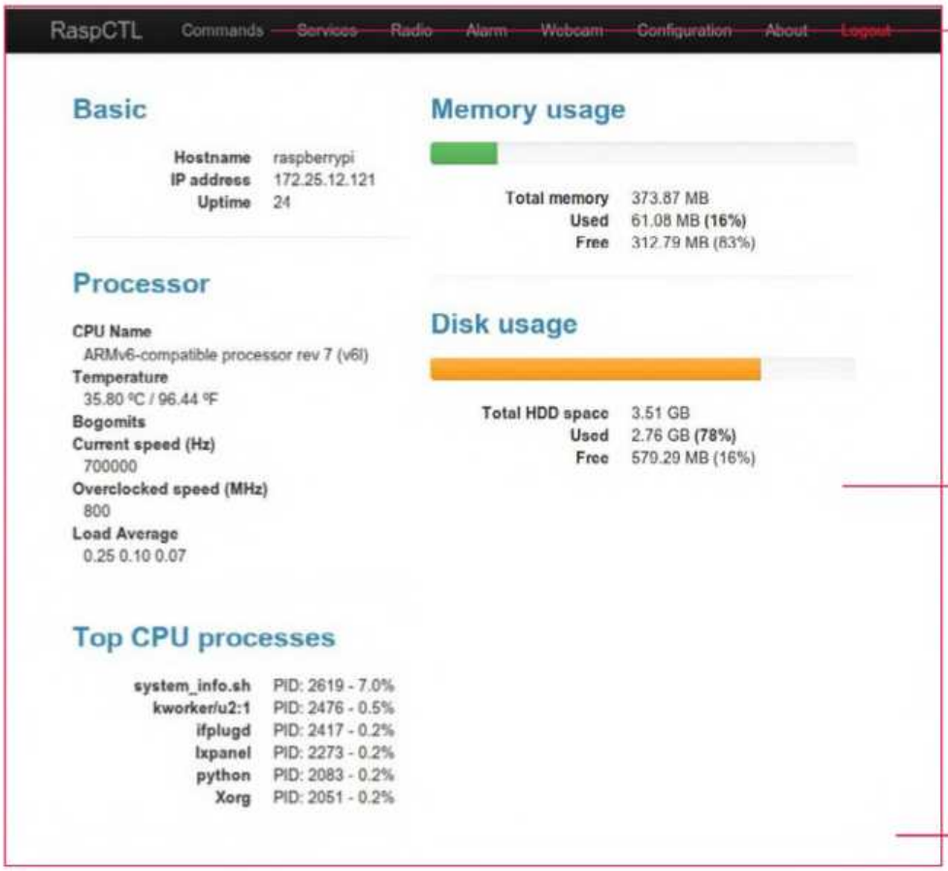
While losing your Raspberry Pi or the data on it, might initially seem like a disaster, don't be disheartened. As long as you have taken steps to backup data or clone your SD card, you at least have continuity when you resume the project. You can also check our boxouts for methods to help you deal with physical theft.

Pocket your Pi

If you're still concerned with your Pi's safety, put yourself in the place of a potential thief. Where would you stash it? Probably in your pocket. The Raspberry Pi is small enough to take with you, so why leave it lying around? Any security questions relating to your Raspberry Pi can be addressed by keeping it close whenever necessary.

Remotely control your Raspberry Pi

Use a web interface to control your Pi and employ it as a fileserver or media centre from a remote location using any web-connected device



Commands Create custom commands for running your Raspberry Pi

Other utilities Seeing through your webcam and setting an alarm are just two additional things you can do with your Pi

Main window Get the full details of the currently running system from the web

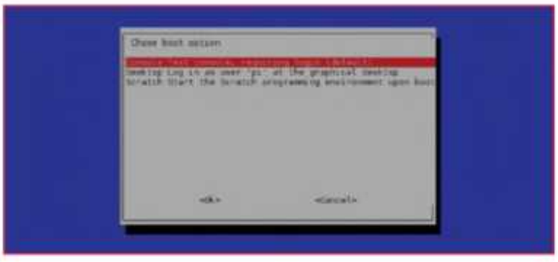
What you'll need

- Raspbian set to command line
- RaspCTL
- Internet connection

Not everyone uses the Raspberry Pi while it's hooked up to a monitor like a normal PC. Due to its size and portability, the Raspberry Pi can be located almost anywhere that it can be powered and it's widely used as a file server, media centre and for many other nontraditional applications. Some of these uses won't easily allow access to a monitor for easy updates and maintenance. While you can always SSH in, it's a bit slower than a full web interface that allows for custom commands and a view of the Pi's performance. We're using software called RaspCTL, which is still in development, but works just fine for now.

01 Update your Pi!

To make sure the Raspberry Pi works as best it can, you'll need to update Raspbian. Do this with a `sudo apt-get update && apt-get upgrade`, followed by a firmware update with `sudo rpi-update`. Finally, if you're booting to LXDE, enter `raspi-config` and change it to boot to command line to save power.

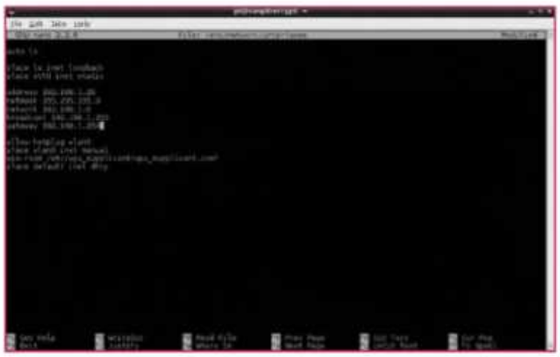


02 Edit the IP

For everything to work more easily, you should set the Raspberry Pi to have a static IP of your choice. To do this, edit the networking config by using:

```
$ sudo nano /etc/network/interfaces
```

...and change iface eth0 inet dhcp to iface eth0 inet static.



03 Set up a static IP

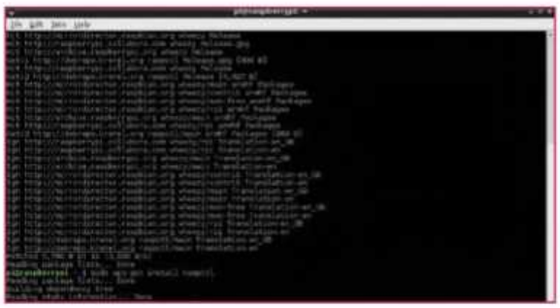
Add the following lines under the iface line with your relevant details:

```
address 192.168.1.[IP]
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.[Router IP]
```

04 Ready to install

You'll need to grab the public keys for the software we're going to install by using the following commands. The first will take just a moment to download the software, while the other quickly installs it:

```
$ wget debrepo.krenel.org/raspctl.asc
$ cat raspctl.asc | sudo apt-key add -
```



05 Add the repository and install

Add the repository to the source's file with the following command:

```
$ echo "deb http://debrepo.krenel.org/ raspctl
main" | sudo tee /etc/apt/sources.list.d/raspctl.
list
```

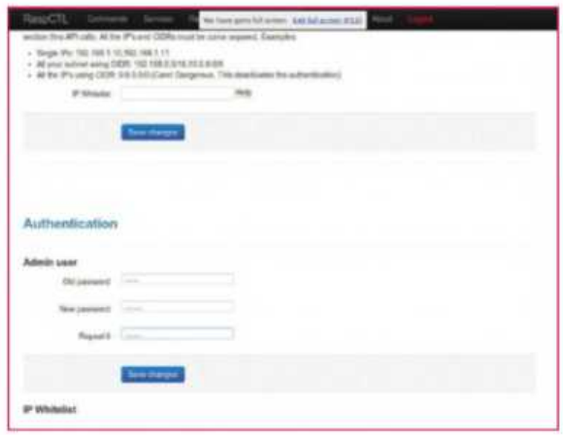
...and finally install the software with:

```
$ sudo apt-get update
$ sudo apt-get install raspctl
```

06 Access your Raspberry Pi

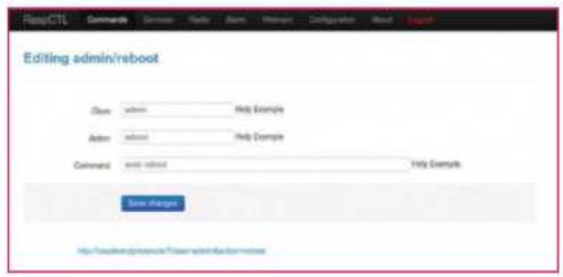
Now the software is installed you can start to access your Raspberry Pi from anywhere on your network. To do this type the following into your address bar, with the IP being the one we set up earlier:

```
http://[IP]:8086
```



07 Change your password

The default username and password is admin for both fields, and you should make sure to change that before doing anything else. Go to Configuration along the top bar and find the Authentication field at the bottom of the page. Input the original password (admin), followed by your new passwords. The username will remain as admin.

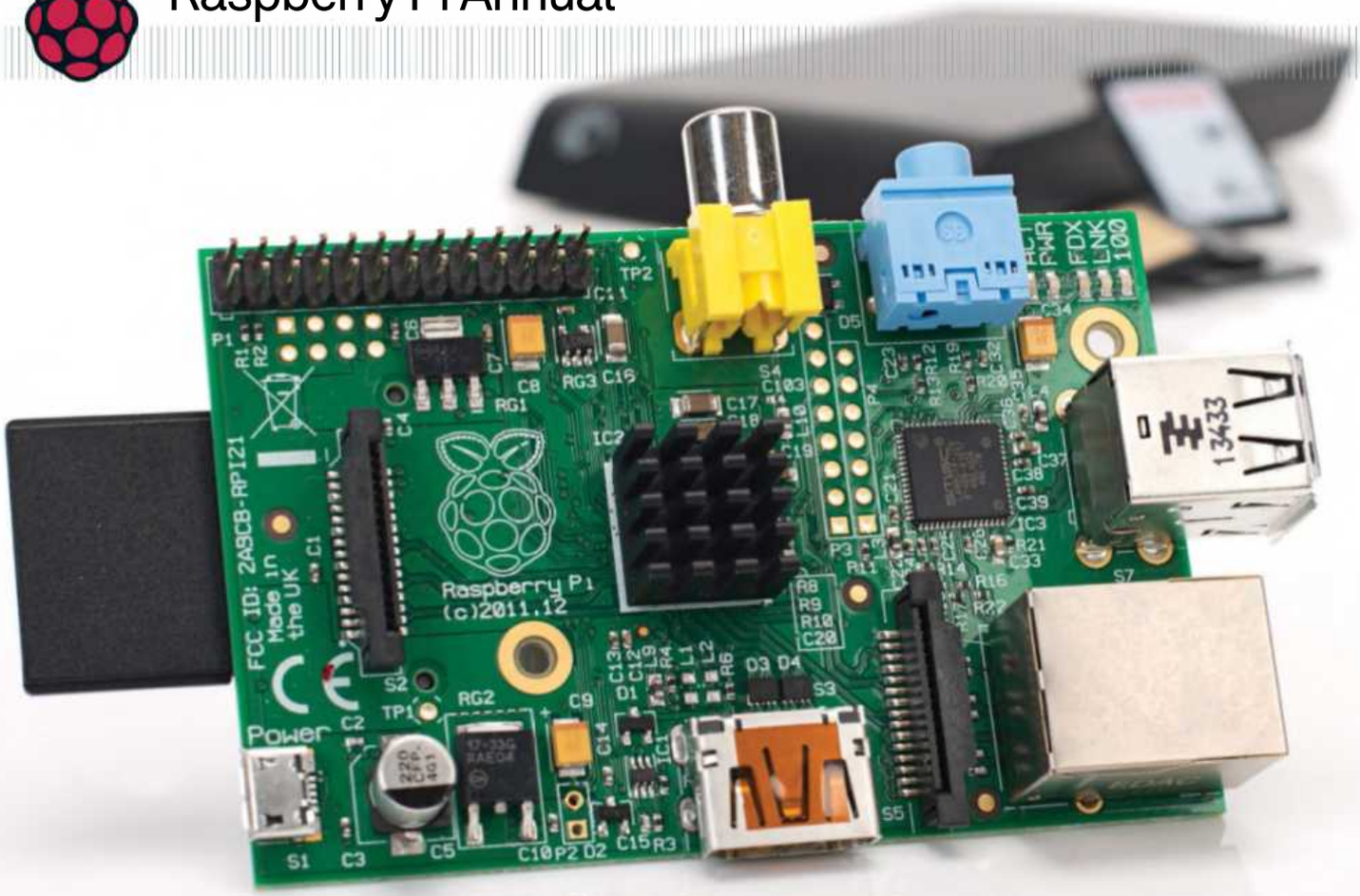


08 First command

Go to Commands on the top bar to begin creating commands to run. Here you'll need to add a class – a user-defined way to filter your commands that won't affect the way it's run – a name for the command and the actual command itself. The commands won't necessarily run from the pi user unless you tweak the config files.

09 More functions

The web interface has a few extra functions apart from running commands, such as the ability to view the webcam and connect to radio services. Updating the software every so often will also allow you to make sure it keeps working. Play around with it and see what best suits you.



Supercharge your Raspberry Pi

Get the most out of your Raspberry Pi with these performance-enhancing tips and tricks

Your Raspberry Pi is plugged in. Raspbian is installed on the SD card and you are right in the middle of setting up a wireless print server or building a robot to collect your mail from your doormat.

But are you truly getting the most from your little computer? Do the components you're using maximise the potential of your Raspberry Pi or are they holding it back? Perhaps you haven't explored the full set of options in Raspbian, or you're running the entire OS from SD card, something that can reduce SD card lifespan.

Various tools and techniques can be employed to improve performance, from choosing the right hardware to overclocking the CPU. You might even maximise storage space on the Raspberry Pi's SD card or all but replace it with a secondary device to help improve speed.

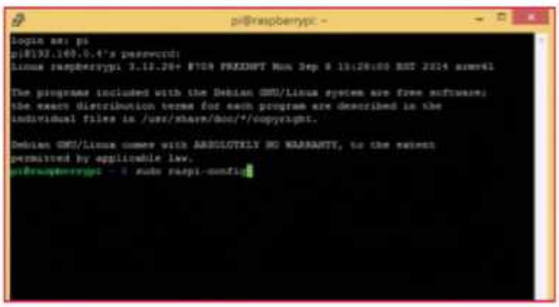
Use these tips and tricks to reconfigure your Raspberry Pi setup and optimise software and hardware to ensure you get the best performance for your projects.

01 Use better storage hardware

Your choice of storage media can have an impact on your Raspberry Pi's performance, regardless of the operating system. A low capacity SD card with poor error correction, is going to be slower than a larger card with greater resilience, so you need to find the right balance for your project and shop wisely.

02 Choosing the best SD card

Various standards of SD card are available, with the more expensive designed for better error correction. For the best performance on your Raspberry Pi, choose an SDHC card with a high rating. The same advice applies to MicroSD cards, which you can use on your Raspberry Pi with an SD card adaptor or directly insert into a Raspberry Pi B+.



03 Make the most of your storage

You'll typically need 1-2GB of storage for your chosen Raspberry Pi distro, so any remaining storage on your SD card will be used for updates and data you create or save.

In Raspbian you can open a command line and run the configuration utility to gain more space (only if your SD card's greater than 2 GB):

```
sudo raspi-config
```

04 Expand the Raspbian partition

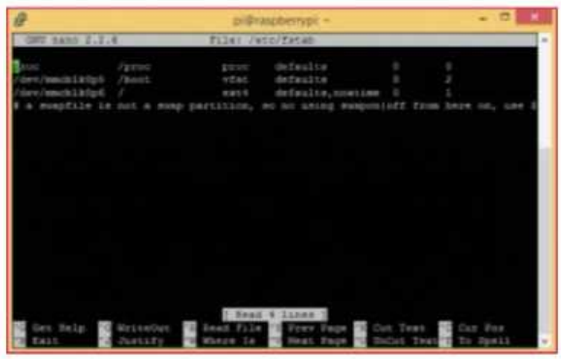
Maximising the partition affords the full capacity of your SD card, which will increase the media's lifespan (there is more space to write too, so the same sectors aren't being overwritten as often).

With raspi-config running, use the arrow keys to select expand_rootfs in the menu. Then wait briefly while the partition is resized.

05 Write data to RAM

Rather than reading and writing data to your SD card – something that will eventually result in a deterioration of reliability and performance – you can configure Raspbian to write to the system RAM, which will speed things up slightly and improve SD card performance.

This is achieved using fstab (file systems table), a system configuration available in most Linux distros.



06 Enable fstab in Raspbian

This is much like creating a RAM disk in Windows and is almost as easy to setup. In the command line, enter:

```
sudo nano /etc/fstab
```

Add the following line to mount a virtual file system:

```
tmpfs /var/log tmpfs defaults,noatime,nosuid,mode=
0755,size=100m 0 0
```

Follow this by saving and exiting nano (Ctrl+X), then safely restarting the Pi:

```
sudo shutdown -r now
```

Above There's a great guide to SD cards at **elinux.org/RPi_SD_cards**

Buy rated SD cards

It's all too tempting to boot up your Raspberry Pi with an image copied to an SD card that you just pulled out of your DSLR or phone. After all, they're all the same, right? The chances are that your chosen SD card was one that you had lying about when you bought your Raspberry Pi. It might be good enough but if you want the best performance, a high-rated SDHC card with plenty of space is recommended. Such media is inexpensive and can be bought online or in supermarkets. Just make sure you're buying a known brand!

Above Having your filesystem on a USB stick is great for backups as well as performance boosts

Picking an external USB drive

Speeding up your Raspberry Pi by migrating the root filesystem to an external USB drive is a start, but what sort of device should you use for the best performance? With a USB thumb drive you can add flash storage up to 16 GB without running into any significant problems (the larger the drive, the greater the current is required to read/write). Anything larger is expensive and unnecessary. If you're planning to use an external HDD, there are no power issues as it will have its own power supply. As ever, your choice should suit your project.

07 Configure fstab for fast performance

Upon restarting, the virtual file system will be mounted and /var/log on the RAM disk. Other directories that can be moved to RAM include:

```
tmpfs /tmp tmpfs defaults,noatime,nosuid,size=100m 0 0
tmpfs /var/tmp tmpfs defaults,noatime,nosuid,size=30
m 0 0
tmpfs /var/log tmpfs defaults,noatime,nosuid,mode=0755,
size=100m 0 0
tmpfs /var/run tmpfs defaults,noatime,nosuid,mode=0755
,size=2m 0 0
tmpfs /var/spool/mqueue tmpfs defaults,noatime,nosuid,m
ode=0700,gid=12,size=30m 0 0
```

Add each to /etc/fstab in nano.

08 Move your OS to a HDD

If you're concerned about the lifespan of the SD card, why not reduce your Raspberry Pi's reliance on it? Instead of using the SD card as a sort of budget SSD, change its role and add a HDD or USB stick to run the operating system, leaving the SD card for bootstrapping. This can give a marked performance boost to the SD card.

09 Back up the SD card

Begin by creating a copy of your Raspberry Pi's SD card. Shut down, remove the card and insert it into your desktop computer. In the command line, run:

```
sudo dd bs=4M if=/dev/sdb of=~/backup.img
```

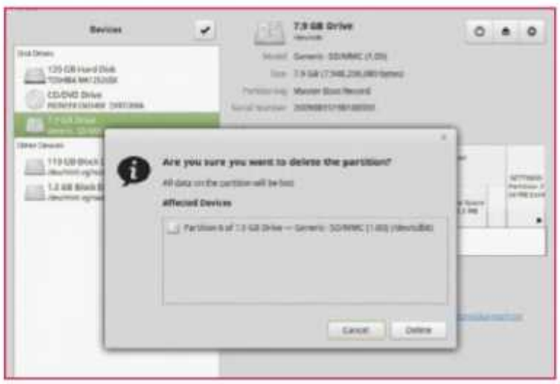
The path /dev/sdb represents the SD card. Copying should take 5-10 minutes. When complete, remove the SD card and connect your USB device.

10 Copy Raspbian to USB

Using a blank Ext4-formatted USB thumb drive (or external HDD) as the destination drive, enter:

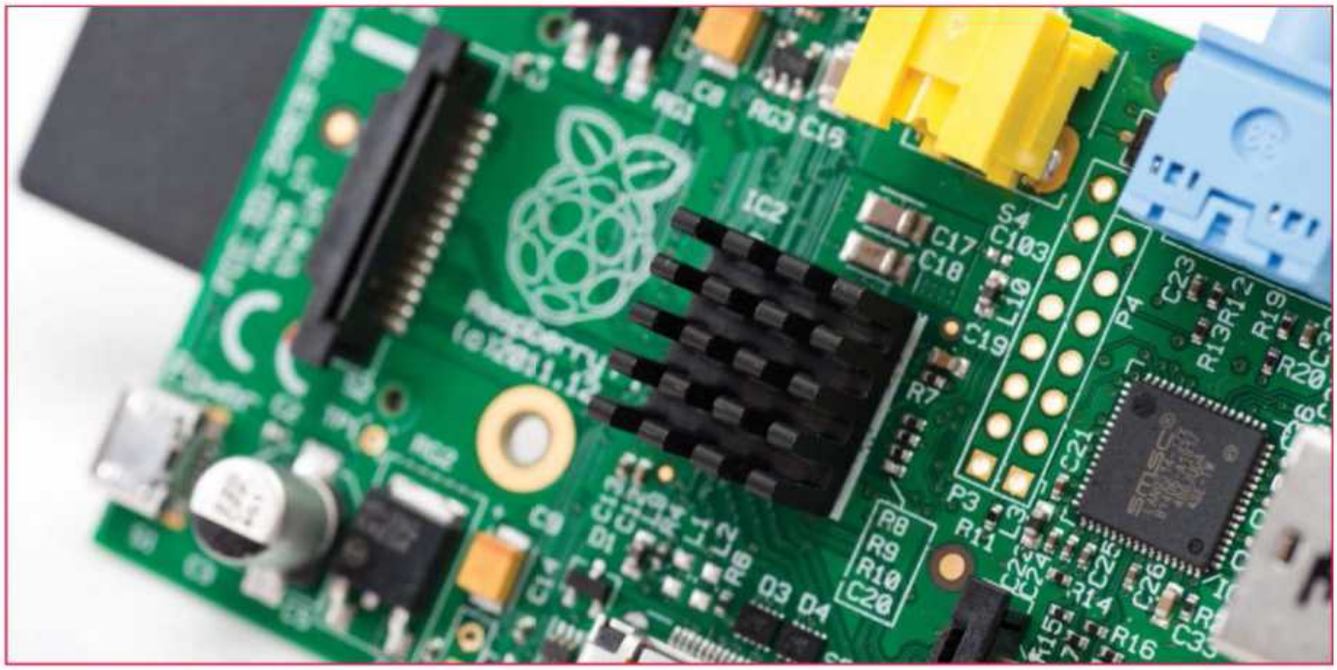
```
sudo dd bs=4M if=~/backup.img of=/dev/sdc
```

Leave the backup on your computer, just in case something goes wrong. With an SD card and USB storage device sharing an identical disk image, it's time to consider what you're going to do next – create a faster Raspberry Pi.



11 Split the Raspbian partitions

Ideally, the boot partition should remain on the SD card while the root filesystem is run from the external HDD or USB thumb drive. Using your preferred partition manager (Disk Utility is in most distros), unmount and delete the root filesystem from the SD card, ensuring you have retained the boot partition. After removing the SD card, connect your USB device and delete the boot partition, taking care to leave the root filesystem intact. Then resize the root filesystem on the USB device, making sure that 10 MB remains.



Above Heatsinks for the Pi are widely available and usually cost less than $10

12 Identify the root filesystem

With this configuration you're going to have the SD card and the external USB storage connected, so you need to tell the Pi where the root filesystem is. Still on the desktop Linux computer with your SD card inserted, run:

```
sudo nano /boot/cmdline.txt
```

Find root=/dev/mmcblk0p2 (or similar) and change that to read root=/dev/sda2 which is your external USB storage. Save and exit.

13 Add other USB devices

You can now restart your Pi with the storage devices attached, but as soon as you connect further USB media you'll suffer problems. Avoid this by installing gdisk:

```
sudo apt-get update
sudo apt-get install gdisk
```

Then run gdisk:

```
sudo gdisk /dev/sdb
```

Enter ? to display the options and select Recovery and Transformation options (experts only), followed by Load MBR and Build Fresh GPT. Tap ? one last time and select 'Write Table to Disk' and exit. Remove and replace the USB device and run gdisk again. This time enter I and then 1 to display the Partition Unique GUID.

14 Make your Pi fast & reliable

Make a note of the GUID and then switch to the SD card. Reopen cmdline.txt and change root=/dev/mmcblk0p2 to root=PARTUUID=XXXXXX, where the numerical string from the partition unique GUID should replace the XXXXXX. When you're done, save and exit. You can then start your Raspberry Pi. Congratulations, your Raspberry Pi is now faster and more reliable to use!

15 Boost performance with overclocking

Need more from your Raspberry Pi? It is possible to overclock the computer, although you should be aware of the risks inherent with this activity. You should also ensure that your Raspberry Pi's processor is suitably cooled – heatsinks for the CPU, Ethernet controller and power regulator can be purchased online.

16 Overclock your Raspberry Pi

Overclocking is available through raspi-config. Launch from the command line and arrow down to the overclock option. Four further options are available: Modest, Medium, High and Turbo. With your ideal clock speed selected, exit raspi-config and restart your Raspberry Pi to apply:

```
sudo shutdown -r now
```

Now you will need to perform tests to see how stable it is overclocked. Raspberry Pi founder, Eben Upton, suggests running *Quake 3* as a good stress test. Should the Pi fail to boot, hold Shift to boot without overclocking, run raspi-config and select a more modest overclock.

17 Run Raspbian without the GUI

Despite these changes, you may find that the GUI remains slow. If you find yourself running a lot of commands in bash, the best thing to do is disable launching into X. In raspi-config, choose boot_behaviour and select the first (default) option to ensure your Pi boots to the command line. Should you need the GUI, enter 'startx' in Terminal.

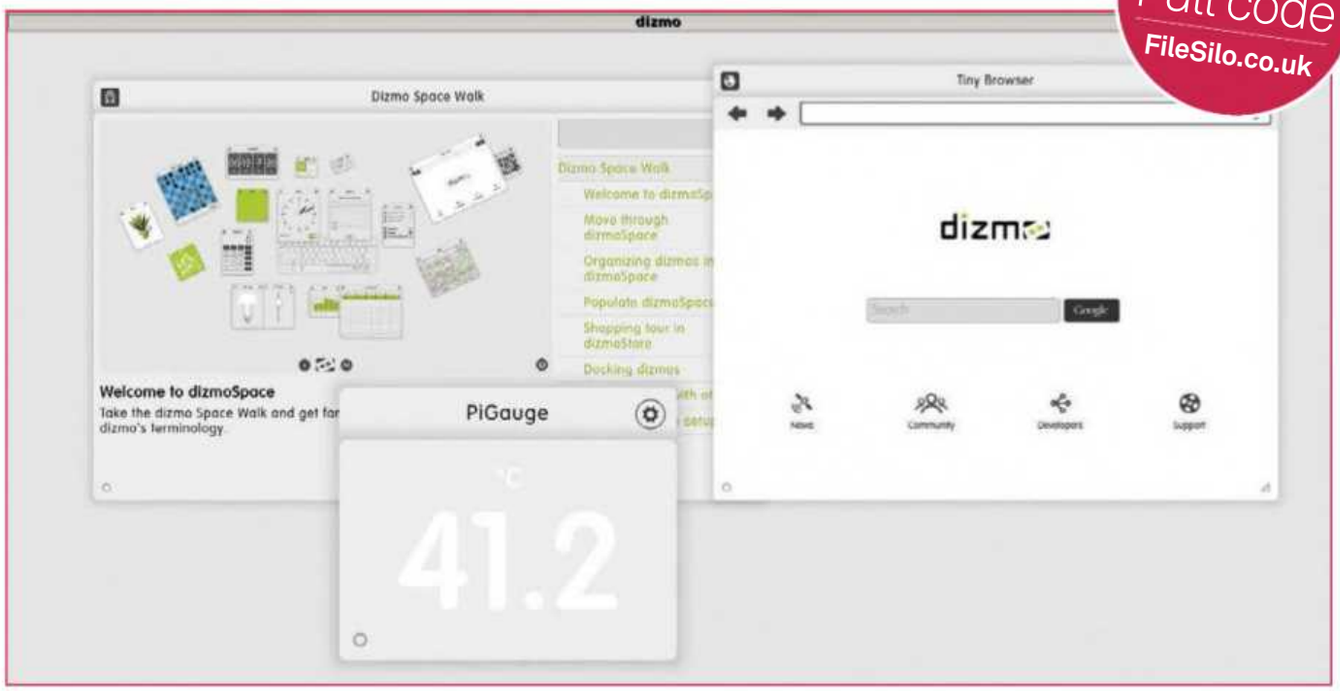
Overclock with a heatsink

Overclocking is potentially dangerous to any computer system, which is why it's great that the Raspberry Pi developers have included the facility in their approved operating system and allowed its use under warranty. If you're using this feature, heatsinks and water cooling systems are available for the Raspberry Pi to ensure you don't bake the CPU and RAM when in use.

Monitor CPU temperature with Dizmo

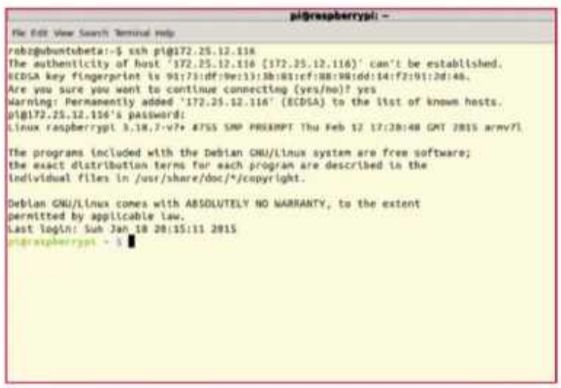
Turn your Raspberry Pi into an Internet of Things with this CPU temperature gauge tutorial

Full code FileSilo.co.uk



The Raspberry Pi is an exciting prospect for people interested in an Internet of Things – size, power and flexibility make it perfect for powering any Internet-connected device around the home or office. Setting up a Raspberry Pi to be the brain of an IoT network isn't exactly a case of selecting the right software in Raspbian, though; there's a lot of custom work you need to do to get one going.

This is where Dizmo comes in, enabling you to control IoT objects using an online API that you can then access remotely. To show you how it works, we're going to have it track the Raspberry Pi's core temperature. In this tutorial we are going to work entirely over SSH, but you can easily do this straight on the Pi – the benefit of SSH though is that for a real IoT, it will be easier to maintain remotely.

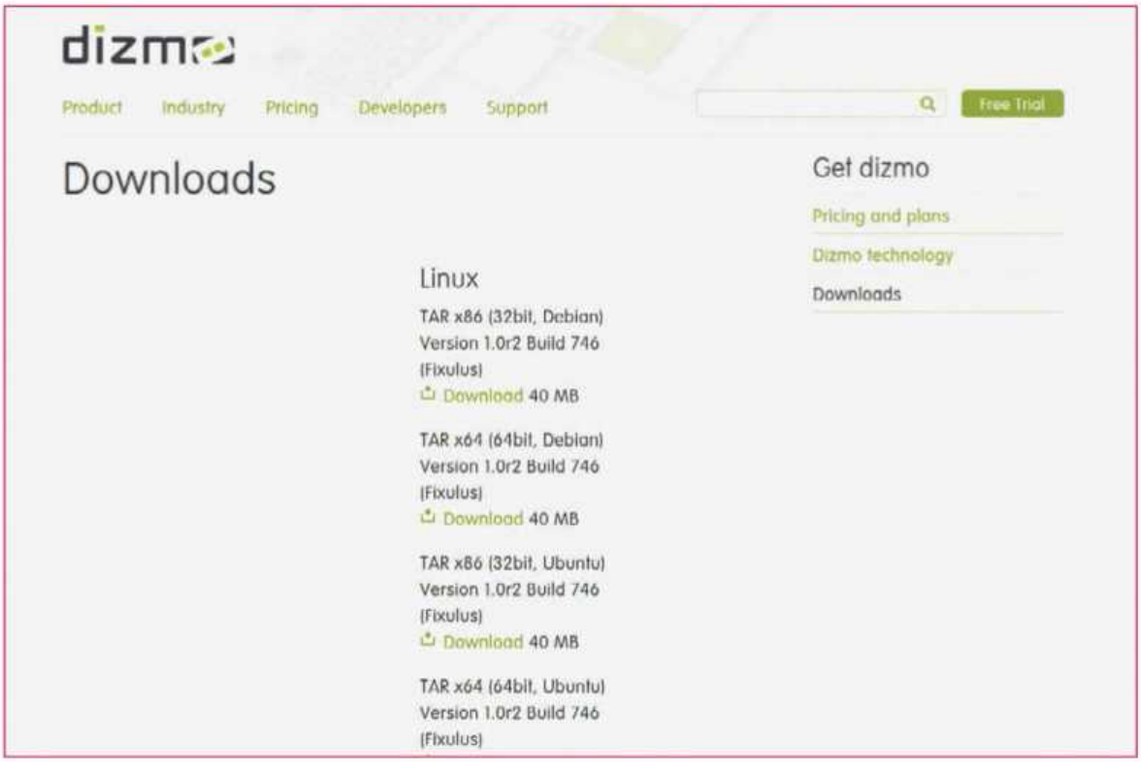


Above Dizmo is designed to be a multi-touch interface

01 Dial into your Pi

Make sure your Raspberry Pi can connect to your network, either via Wi-Fi or ethernet cable, and find out the IP address by using **ifconfig**. Use this IP to dial into the Pi from another system with:

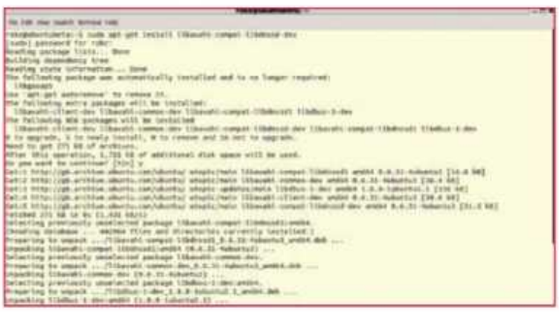
```
$ ssh pi@[IP address]
```



Left Builds are available for various distros on the Download page, and you can also check the pricing plans

02 Install dizmoSpace

If you haven't already, head to **www.dizmo.com**, grab dizmoSpace and install it to the system you plan for it to work with. All you need to do is download the zip and unpack it, then click the Dizmo icon or run it from the terminal.



03 Launch issues?

If Dizmo is complaining about libraries when you try to run it, you'll need to install some extra software. Open the terminal on the PC you're working from and install the extra software with the following:

```
$ sudo apt-get install libavahi-compat-libdnssd-dev
$ sudo apt-get install libavahi-client-dev
```

04 Download node.js

Now, we need to grab the latest version of node.js for the Raspberry Pi. Back in the SSH connection to your Raspberry Pi, use the following:

```
$ sudo wget http://node-arm.herokuapp.com/
node_latest_armhf.deb
$ sudo dpkg -i node_latest_armhf.deb
```

A Dizmo widget is a HTML file, packaging resources together to create an interface or graphic. Our HTML file uses jQuery

05 Add framework

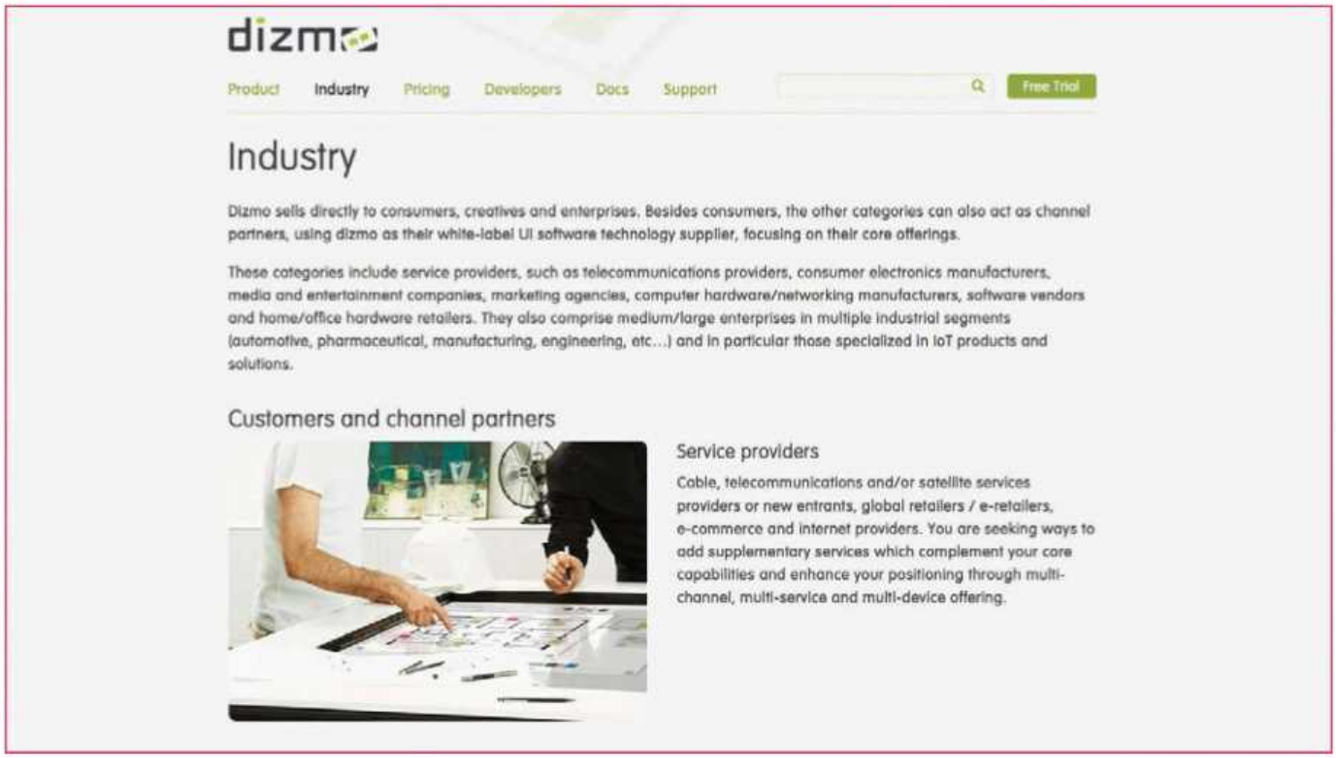
Use **node -v** to check if it's installed correctly – it should spit out a version number for you. Once that's done, install express.js, which will be our web application framework:

```
$ sudo npm install -g express
$ sudo npm install -g express-generator
```

06 Install framework

We'll create the folder www in var and create a symlink for everything to run. Do this by moving to var, creating www and making the symlink with:

```
$ cd /var
$ sudo mkdir www
$ cd www
$ sudo ln -s /usr/local/lib/node_modules/
/node_modules
```



Above As it's multi-touch, Dizmo is perfect for interactive table displays in meetings

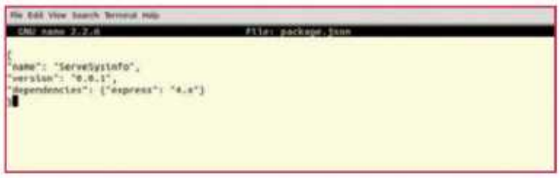
Internet of Things

It's not a very descriptive term, but the Internet of Things can be almost anything. Any item that is or can be connected to the internet or networks, such as modern automated lights, can be connected up to Dizmo and the Raspberry Pi.

Dizmo space walk Enjoy some pre-installed projects to see exactly what Dizmo can do

PiGauge Create a custom app to monitor the temperature of your Raspberry Pi, and then go even further

Browser Create an entire custom display using a variety of information that can connect to and through the Pi



07 Package file

First, create the file package.json with `sudo nano package.json`, then enter:

```
{
"name": "ServeSysinfo",
"version": "0.0.1",
"dependencies": {"express": "4.x"}
}
```

08 App node

Now, create a file called app.js and enter the following:

```
var express = require('express');
var app = express();
app.use(express.static(__dirname + '/public'));
app.listen(3000, function(){
console.log('listening on *.3000');
});
```

09 Start node.js

You can now start the node server by typing in:

```
$ node app.js
```

It will say it's listening on *.3000. Start up a new terminal, **ssh** in, and create the folder /public with **mkdir /public** to save all of the CPU data in.

10 CPU information

We are going to use the **vcgencmd** command to get the CPU information from the Raspberry Pi. We will write a script that will do this and then write the info to sysinfo.json. Download the file grabsysinfo.sh from FileSilo and put it in /usr/local/bin.

11 Make a cronjob

We will make it so that the temperature is updated every ten minutes. You can make it update much faster if you want, but have a play around with that. Open up cron with `sudo crontab -e` and add this at the end:

```
*/10 * * * * /usr/local/bin/grabsysinfo.sh
```

12 Start creating the widget

It is time to actually start building the widget. First of all, create a folder on your local machine called Gauge and **cd** to it. Now you need to download the first file called info.plist into here by using the following:

```
$ wget x/info.plist
```

13 Index file

A Dizmo widget is basically a HTML file, packaging resources together to create an interface or graphic. Here, we have the main HTML file that uses jQuery, which helps display the temperature. Still in the Gauge folder, download it with:

```
$ wget x/index.html
```

> With these building blocks, you can now start doing more interesting IoT things – controlling the GPIO ports, getting more information

14 Style guide

Now we'll add the CSS style sheet for the Dizmo widget. As usual, this styles up the display on the page that will become our widget. Download it with:

```
wget x/style.css
```

15 Final application

The final step is to create the application.js file, which will call the temperature from the Raspberry Pi using Ajax. You can download it using:

```
wget x/application.js
```

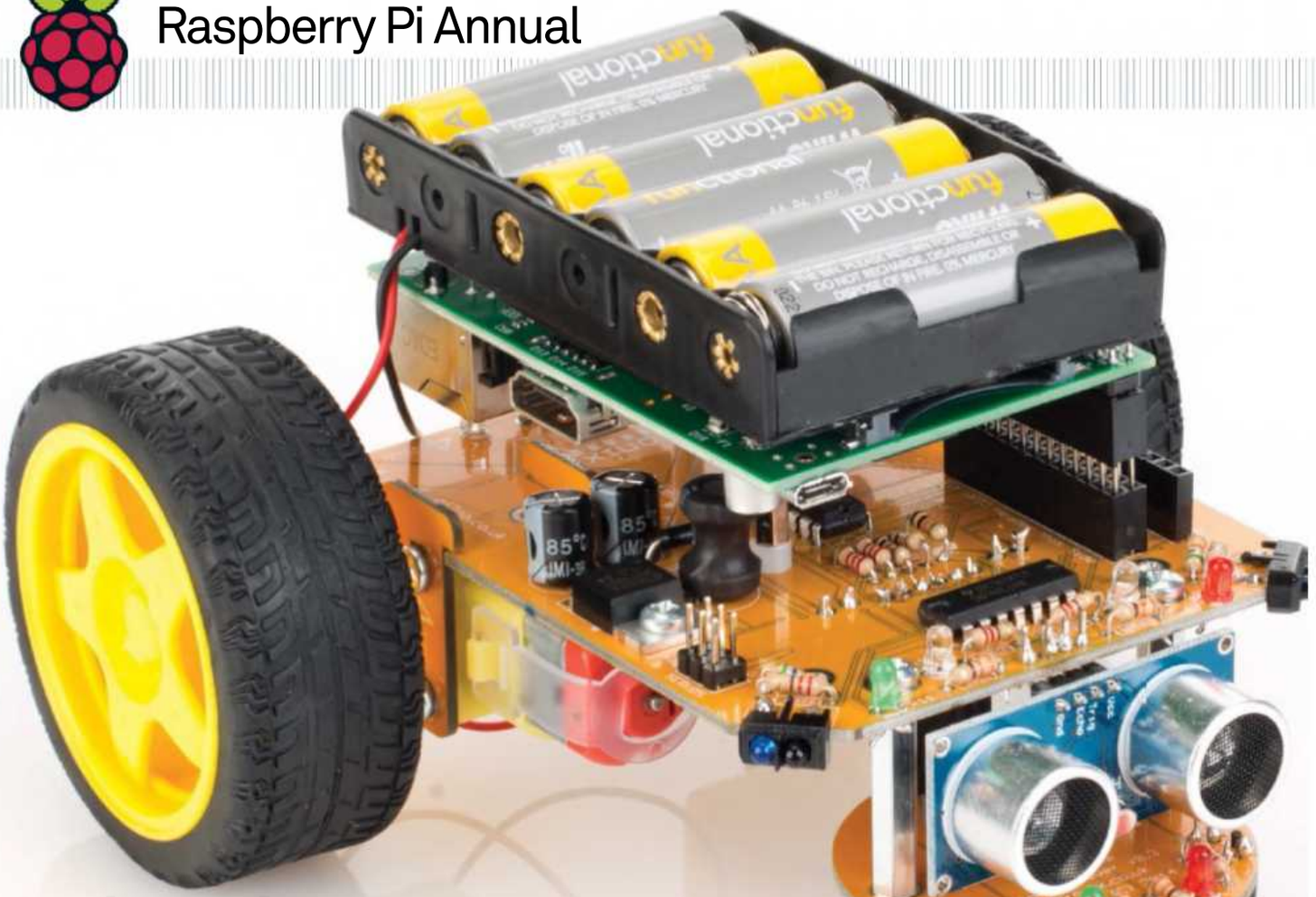
Change the IP address to the one on your Pi.

Once that's done, you can test out the widget – compress the Gauge folder to a .zip and then change the .zip to a .dzm. Launch dizmoSpace and drag the dzm file onto it for it to start.

16 Get coding

With these building blocks, you can now start doing more interesting IoT things – controlling the GPIO ports, getting more information, having it connect to other objects to control them as well. Check out the Dizmo website for more details on projects that you can do.

Above We've gone for a simple CPU temperature gauge, but the possibilities really are endless



What is a Raspberry Pi robot?

Is the Pi robot a specific product or just a concept? An easy answer for some, but not everyone knows the score

So you have some Raspberry Pi robots every now and then in the magazine. This may sound like a stupid question but... what is a Raspberry Pi robot?
It's quite simply a robot that is powered, or can be powered, by a Raspberry Pi. Nothing too complicated to it – well, apart from actually making the robot and getting it to talk to the Pi. That can be very complicated.

So it's not a special model of the Raspberry Pi?
No, it's not a Model Bot Raspberry Pi or whatever they would call it. Just a plain Raspberry Pi Model B or B+, connected up to a robot chassis via the usual mount points or with a bit of sticky tack or glue depending on how cavalier you're feeling.

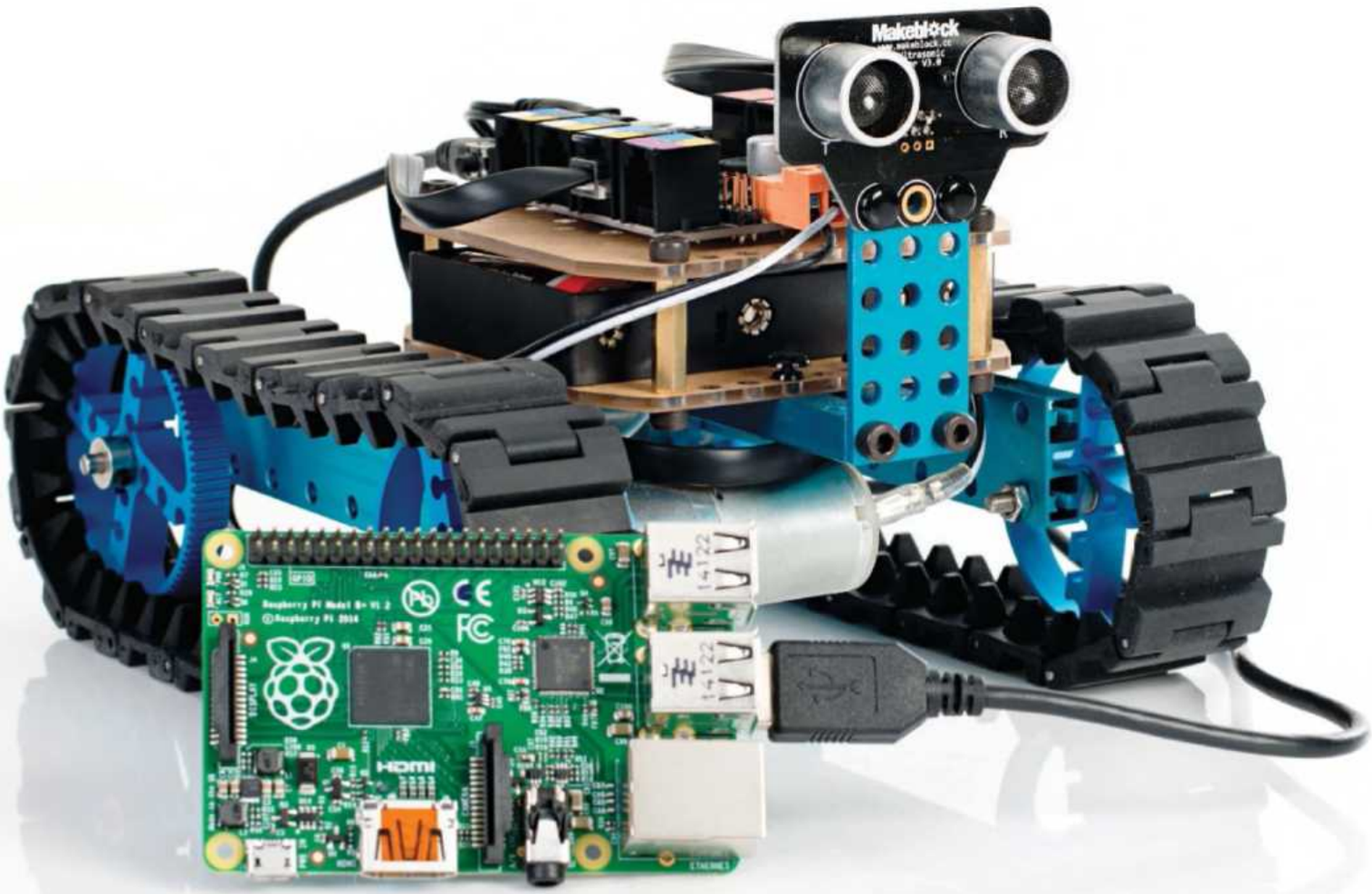
What about the compute module?
Yeah the compute module works as well – in fact it can work a lot better as it has 200 GPIO pins when connected to the I/O board, compared to the 26 or 40 of the other Raspberry Pi models. Although with a bit of know-how and a fair few components you can get it to run a robot without actually needing the I/O board.

So the GPIO port is what powers these robots then?
The pins allow you to read data and activate circuits and motors, which is just about everything you need to do on a robot. You could use the camera port and USB ports for things like a video feed or connecting plug-and-play devices. However, the majority of what you will be able to make a robot do will be via the GPIO.

Can these robots use anything else to power them?
Sure, some are pure Raspberry Pi, some are a mixture between Pi and Arduino and others can still be controlled by either a Pi or with an Arduino device like an Uno or Leonardo. The Arduino stuff is usually good for controlling servos as that's what it's designed for, whereas the Pi has more processing capabilities for analysing data.

Can all Arduino robots be controlled by a Pi and vice versa?
With enough wire, components and patience

Below Makeblock is another Arduino robot that's programmable on the Raspberry Pi – check out our tutorial on doing this: **bit.ly/1y8cB7i**



it probably can be, but there are varying levels of difficulty in that. Some of the robot parts will require more control than others, for example. The easy answer is no, not really.

Can any robot be powered by a Raspberry Pi?
That's a similar answer to before really. The Raspberry Pi has a lot of connectivity options and there are a few robots we've seen that have been hacked to use the Pi instead of their original intended hardware. One of the limits you start to get is processing power on some robots.

Why is that a limit?
The more sophisticated the robot, the more inputs and input processing is needed. There's also the hard limit on things to output to because of limited GPIO pins. We do have some quite smart robots this issue, so the limit on the Pi isn't too low.

That's convenient.
Very!

What do I need to get started with my very own robot then?

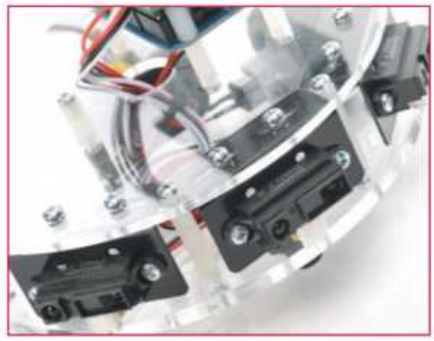
> Arduino is usually good for controlling servos as that's what it's designed for

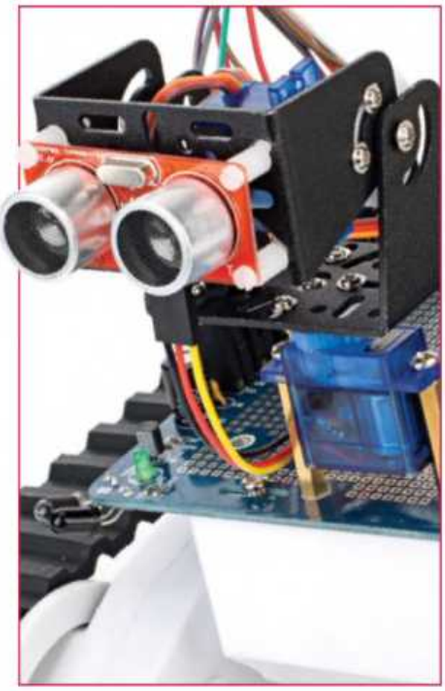
Well kits are usually the best idea and we cover a lot of those in our big robot feature this issue. We have small, cheap ones and bigger, expensive ones that can do some really cool stuff.

All of them involve varying levels of difficulty while building. Check out our feature starting on the next page to find out which ones might suit you and head over to the websites for more info.

Our top Raspberry Pi robots

Discover our favourite robotics kits and learn to program them with your Pi





The rise of our robot overlords is well underway – give it another five years and we'll all be watched over by Pi-powered machines of loving grace. In the meantime, though, we've rounded up the very best DIY robotics kits available to buy right now that are designed to work with your Raspberry Pi, so you can get a head start on the inevitable revolution. Whether they're Arduino or Raspberry Pi-based, we're getting all of our robots to listen to our master Pi controller and showing you how to do the same with your kit. We'll also be scoring these robotics kits to identify their strengths and weaknesses in terms of their build quality, functionality out of the box, the construction process and of course their programmability, to help show you which kit is right for you and where you can get hold of your own.

And what then? Well, we thought we'd put our robots to the test with a series of challenges inspired by the Cambridge Raspberry Jam's Pi Wars event (**www.piwars.org**). Super senpai Rob Zwetsloot will be your compère for a weekend of a robot-building mayhem, and he has cut code capable of rendering these robots skillful enough to breeze through a line-following challenge, a proximity alert test, a tricky obstacle course and a three-point turn examination. Not content to stop there, though, Rob also reveals how he got one of our robots to play a damned fine round of golf (for an automaton) and another two to battle each other (sumo style).

So it's time to introduce you to our hand-picked team of robots – some of whom you might recognise from issues of **Linux User & Developer**. Over the next few pages you'll meet Rapiro, our most humanoid and delightfully articulate Gundamesque robot; GoPiGo and Pi2Go, two nippy little two-wheel tricars with ball-bearing casters for stability at the rear; Frindo, the sensor-loaded, open source mobile robotics platform; Rover 5, the rugged two-track tank with a Seeeduino brain and an inexorable top speed of 1km/s; and Hexy, the six-legged, crab-walking, Thriller-dancing (**bit.ly/1lj2CqR**) force of robotic nature.

Rover 5 Seeeduino

A relative monstrosity, the Seeeduino is fully kitted out and makes a great gift

Technical specs

Manufacturer
Dawn Robotics

Height
170 mm

Width and depth
225 x 235 mm

Weight
1.05 kg

Power
9 volts from 6 AA batteries

Control board
Seeeduino Arduino (ATmega 328P)

Form of locomotion
Two treads powered by four motors

Sensors
Ultrasonic and four corner-mounted infrared sensors

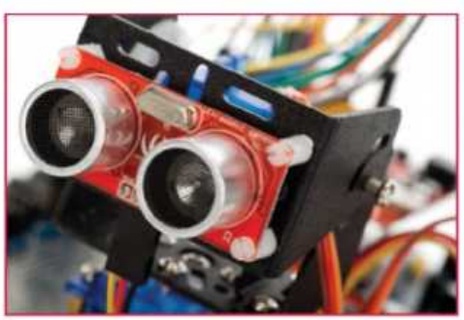
Website
www.dawnrobotics.co.uk

If you recall issue 132, where we made our first robot, you'll remember that the kit we used to build it was from Dawn Robotics – the Rover 5 is sort of a successor to that kit. The Rover 5 is obviously a lot larger and generally has a few more functions than that particular Raspberry Pi robot. Said Raspberry Pi is not needed for the Rover 5 as it is fully powered by the Seeeduino, another ATmega 328P.

Construction is not the easiest and requires an extra hand at times. There's no soldering involved but there are an enormous amount of wires that connect up the board. Couple this with some extremely fiddly nuts and bolts, a manual that is sometimes a bit unhelpful, the odd cheap screw and you get a few problems that take a bit of lateral thinking in order to find a solution. The whole kit is a mixture of different components manufactured separately, which explains some of the discrepancies in the screws and how cobbled together the entire thing actually is.

The big board sits on top of the Rover 5 and is quite well suited for the kit, but it does contribute to the DIY, mashed-together look of the Rover 5 with all the wires flying around.

Programming it is slightly harder than other robots due to using pure Arduino rather than having serial commands or Python functions. You'll need to get right into the code to start programming, however there are some preset tutorial scripts that give pointers on how to create your own code. With the sensors on each corner of the Rover 5, the board and robot can react to almost any obstacle thanks to their coverage, not to mention the ultrasonic sensor also attached to it.



Above The ultrasonic sensors enable the Rover 5 to sense distance

CHALLENGE 1...

Obstacle course

No tires or drill instructors, but the robot still needs to navigate a maze of challenges

Get the code
bit.ly/1uQzsNa

Kept top secret by the Pi Wars organisers, at the time of writing we can only guess at what kind of robot-destroying tasks are planned. The challenge they have set is for remote-controlled bots, but we're going to change the rules slightly and rely a bit more on automation. In our scenario, we're using a walled maze that runs a specific yet random course that the robot needs to navigate without any kind of lines to guide a robot with line sensors. It's a little more tailored to the Rover 5's capabilites, but how so?

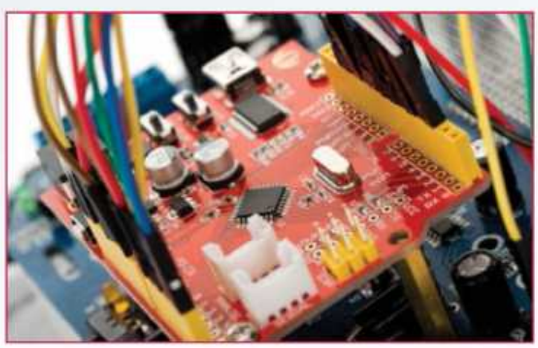
In this challenge, the Rover 5 is perfectly equipped to handle pretty much any course we can throw at it. Thanks to its array of proximity sensors, it will know if there's a wall in the way around its body. We can also use the ultrasonic sensor to figure out its distance from the wall and the way that the wall will change as you travel along the course.

There's some great code for the Rover 5 that allows you to follow a wall along the right of the robot. You can grab it here with the URL **bit.ly/1vp2LLZ**. Upload it via Arduino; it's a bit of a long task, but we will help you out by explaining some parts of it here. Firstly, there are a lot of setup integers created to begin with. These include defining minimums for speed, wall proximity and defining the sensors in general.

Next, we have a small part of the sensing code. All the readings are taken and turned into useful data for the rest of the code – in this case, integers. After that, we have one of the various movement sections. This is used to just move forward, looking to see where an obstacle/wall will be and beginning the chain of events that occurs after noticing or bumping into a wall. This will include backing up, turning left and turning right.

Finally, the code ensures that the sensor keeps an eye on the wall as it runs along it to make sure it's still there.

> The Rover 5 is perfectly equipped to handle any course



Above The main control board connects to the rest of the robot and is easily accessible to add more components

Code listing

```
const int NUM_IR_SENSORS = 4;
const int IR_LED_PINS[ NUM_IR_SENSORS ] = { A0, A0, A1, A1 };
const int IR_SENSOR_PINS[ NUM_IR_SENSORS ] = { A3, A2, A4, A5 };

...

float ultrasonicRange = gUltrasonicSensor.measureRange();
    gRoverIRSensors.takeReadings();
    int frontLeftIR = gRoverIRSensors.lastFrontLeftReading();
    int frontRightIR = gRoverIRSensors.lastFrontRightReading();
    int rearLeftIR = gRoverIRSensors.lastRearLeftReading();
    int rearRightIR = gRoverIRSensors.lastRearRightReading();

...

case eRS_DrivingForwardsLookingForWall:
        {
            // Check to see if we've hit an obstacle we didn't see
            if ( gLeftMotor.isStalled()
                || gRightMotor.isStalled() )
            {
                enterBackingUpState();
            }
            else
            {
                // Check to see if we've found a wall
                if ( ultrasonicRange <= CLOSE_ULTRASONIC_RANGE
                    || frontLeftIR >= CLOSE_RANGE_IR_VALUE
                    || frontRightIR >= CLOSE_RANGE_IR_VALUE )
                {
                    enterTurningLeftState();
                }
            }

            break;
        }

...

void enterFollowingWallOnRightState()
{
    // Point the ultrasonic sensor to the right
    gPanAngle = LOOK_RIGHT_PAN_ANGLE;
    gTiltAngle = LOOK_RIGHT_TILT_ANGLE;

    gPanServo.write( gPanAngle );
    gTiltServo.write( gTiltAngle );

    gLeftMotor.clearStall();
    gRightMotor.clearStall();

    gLeftMotor.setTargetRPM( BASE_WALL_FOLLOWING_RPM );
    gRightMotor.setTargetRPM( BASE_WALL_FOLLOWING_RPM );

    gStateStartEncoderTicks = gLeftMotor.getLastMeasuredEncoderTicks();
    gStateStartTimeMS = millis();
    gRobotState = eRS_FollowingWallOnRight;
}
```

Pi2Go Lite

One of the smallest robots in our test, yet the Pi2Go has a few tricks

The Pi2Go Lite is a very interesting little bit of kit. Coming in a tiny little box and utilising no chassis, in favour of construction via its PCBs, you'd think it would be a super simple robot that follows commands and doesn't really react much to the environment. It makes it sound like a remote control novelty more than anything else. That couldn't be further than the truth, as the Pi2Go is probably the most featureful robot in this feature.

All this functionality comes at a price though, as it's the only robot that requires a lot of soldering and pre-preparation before construction. You'll need to be a bit handy with a soldering iron to do it, although you don't need to strip any wires and such. There are about 50 components to fit, possibly more, and it can be a little time-consuming. The instructions are not extremely helpful, but the individual components are actually listed on the PCB as a rough guide to where things should be fitted. Once the soldering is done though, you just need to put the few parts together to complete it. The website lists a 90 minute construction time, but we found it took somewhat longer – it was no longer than any of the bigger or more complicated robots on the other pages though.

It's a sturdy, compact little thing and it's powered purely by the Raspberry Pi via a custom Python library. Sensing, turning on the LEDs, activating the motors and other physical functions have their own corresponding Python function. It lets you create scripts that can make it fully autonomous, as long as the autonomy only requires distance, line and proximity sensing to operate. At least it can take extra timed or web information from the Raspberry Pi if that's set up correctly.

For the price, functionality and relative ease of programming, it's a fantastic piece of kit that's great for getting into starter-level robotics and slightly beyond. Some soldering skills required though.

Technical specs

Manufacturer
4tronix

Height
90 mm

Width and depth
130 x 145 mm

Weight
0.40 kg

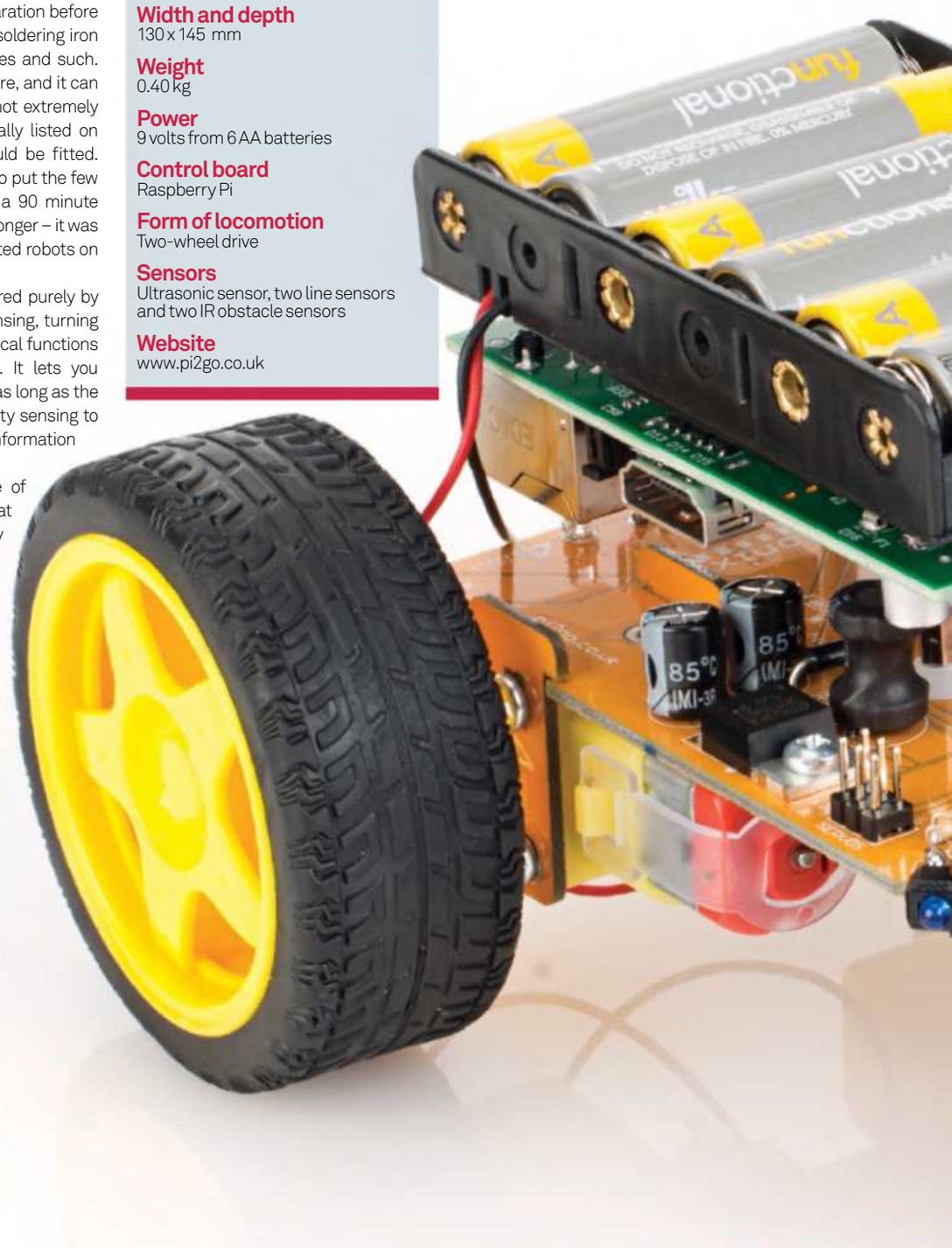
Power
9 volts from 6 AA batteries

Control board
Raspberry Pi

Form of locomotion
Two-wheel drive

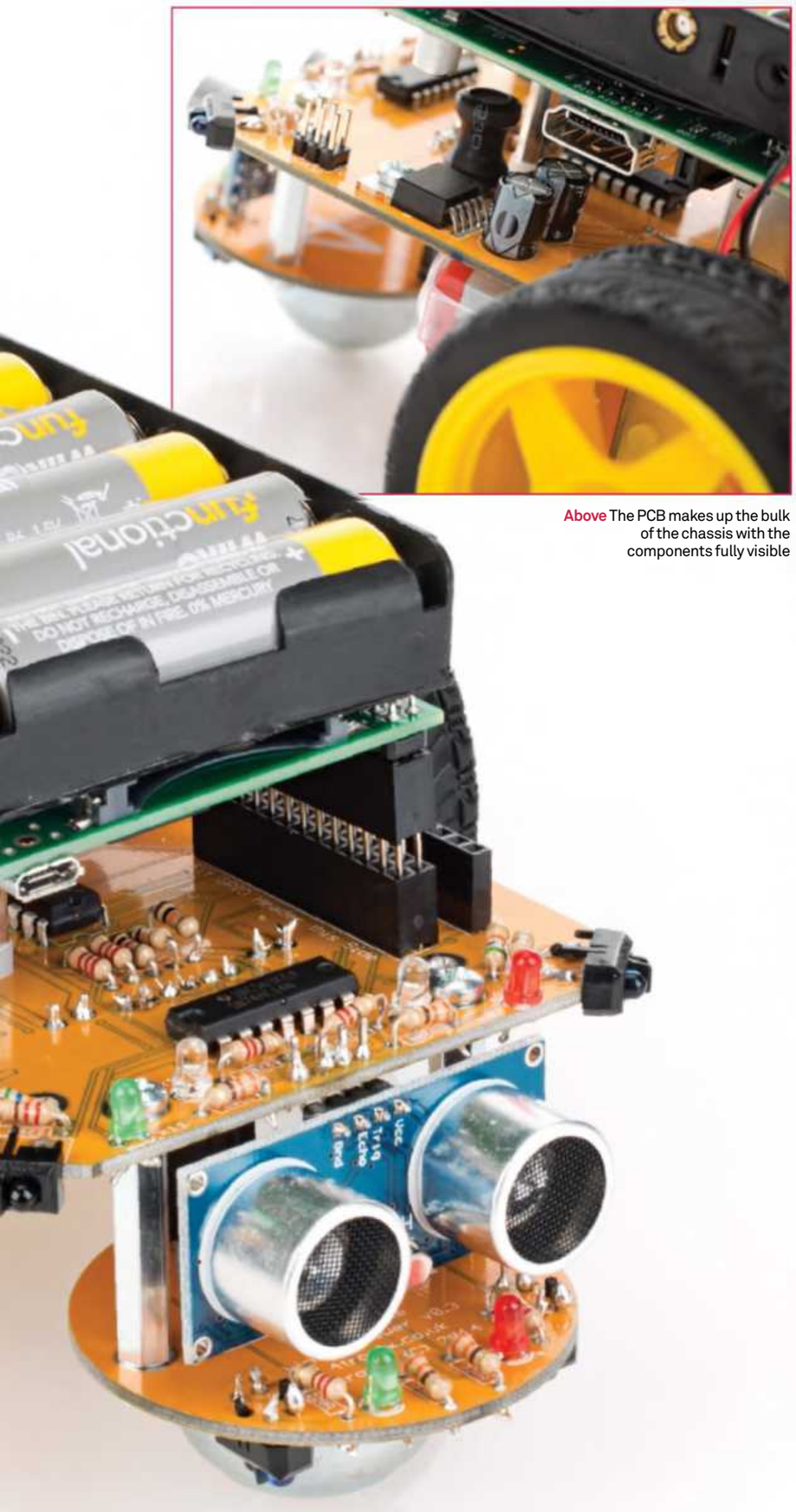
Sensors
Ultrasonic sensor, two line sensors and two IR obstacle sensors

Website
www.pi2go.co.uk



CHALLENGE 2...

Line following



Above The PCB makes up the bulk of the chassis with the components fully visible

Follow the black painted line, although you may not find a wizard at the end of it

Line following is very easy: you put a line on the floor and you expect the robot to follow it. This includes turning as well, following a course accurately to its destination or to accumulate laps. The Pi2Go Lite is the only robot we're looking at this month which comes with line-following sensors, although it is the main unique feature. Sounds like it should be quite simple then, however there's no line-following function in the Python script so we need to build a script for it.

As we said, the solution involves the line-following sensors – these are IR sensors located on the underside of the smaller PCB where the caster sits. We'll assume we've placed the Pi2Go down on the line and you want to go straight forward along it.

One of the problems we're going to run into is that the motors will likely not run at the exact same speed – with a bit of trial and error you can maybe fix this in the first forward command, but for now we'll keep it at 50, which is 50 per cent of its full speed. You can tweak this to be faster or slower as you see fit.

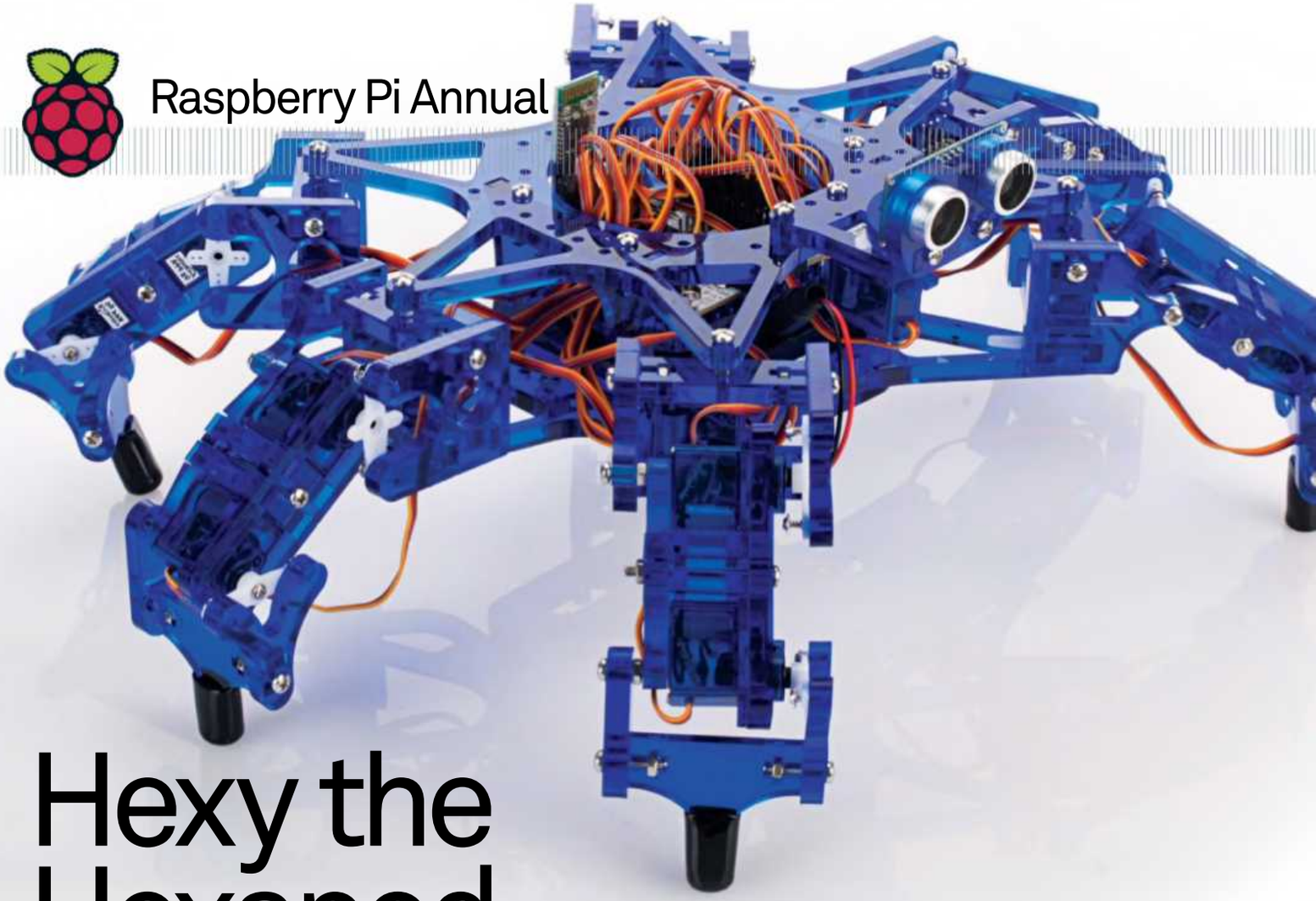
The loop is quite simple: it sees if any of the line sensors are activated. As we're assuming that the line is under the caster wheel, we'll need to correct course in a specific direction for each true statement. You can set the individual speed of the motors (left and then right in the turnForward function), and then we have it pause a bit before returning to full speed.

The code ends when you stop it and cleans up the GPIO port settings before exiting. The code requires the pi2go Python files, which you can grab here: **http://4tronix.co.uk/blog/?p=475**.

Code listing

Get the code
bit.ly/1z1REHW

```
import time, pi2go

pi2go.init()

pi2go.forward(50)

try:
    while True:
        if pi2go.irLeftLine() = True:
            pi2go.turnForward(45, 50)
            time.sleep(2)
            pi2go.forward(50)
        elif pi2go.irRightLine() = True:
            pi2go.turnForward(50, 45)
            time.sleep(2)
            pi2go.forward(50)
     else:
            time.sleep(0.5)

except KeyboardInterrupt:
    print

finally:
    pi2go.cleanup()
```



Hexy the Hexapod

The Kickstarter success story with six legs, 19 servos and some mad dance moves

Technical specs

Manufacturer
ArcBotics

Height
100-140 mm

Width and depth
300-400 x 200mm approx (depending on stance)

Weight
0.90 kg

Power
6 or 7.5 volts from 4 or 5 AA batteries

Control board
Arduino

Form of locomotion
Legs x6

Sensors
Ultrasonic sensor

Website
www.arcbotics.com

We were really impressed by this all-in-one kit that lives up to its Kickstarter promise of being easy to assemble for people of any skill level, including absolute beginners. Everything is neatly packaged in the box and there's even a tiny screwdriver – meaning you don't need any other tools (though to be fair, those servo horns ended up breaking ours, but more on that later).

For the most part the instructions are excellent but there were a couple of occasions where a slight lack of clarity meant that we just followed the images instead, though they were generally spot-on and very useful. You can really see the thought that's gone into it, from the strong and lightweight plastic material to their razor-sharp design. The wiring instructions are perfect and the software tutorials are really useful – you can get an Arduino IDE set up and also dive straight into PoMoCo, ArcBotics' position and motor controller software that's already preloaded with actions (including dance moves) for you to play with.

There's only one real criticism of this kit – the screws are all wrong. There is a big bag of various size screws provided but you don't even use a quarter of them, instead being forced to open each individually packaged servo and borrow the medium screws from them instead because the small holes on the servo horns are far too tiny for the recommended medium screws. The slightly smaller ones from the servo packs fit, so we used those, but you still have to widen those pinholes with brute force. It brings the otherwise speedy build process to a total halt, but all in all, we have to say that Hexy is absolutely worth the trouble.

Above The bluetooth sensor on the Hexy provides an easy way to connect wirelessly

CHALLENGE 3...

Three-point turn

Our robots can go in reverse, but how easily can they do a 180 turn?

This is usually a tricky challenge for robots, especially if it has to be done autonomously like in Pi Wars. The challenge requires the robot to walk out of a designated area and travel just over two metres before performing a three-point turn in an area only 750mm deep. Once it's completed the manoeuvre, it must then return to the starting area. To do this classically, you'd need to know the speed and distance of your robot and travel with extreme accuracy to make the 180 degree turn easier.

The Hexy has an advantage in that it can literally spin on the spot, or at least shuffle its way around. There's even example code to make it turn. All you need it to do is walk to the desired location, turn around and walk back. To do this we made a very simple script where the Hexy walks a few 'steps' forward before attempting a full 180 degree turn and doing the same number of steps back to its starting position. We'll go over some parts of it here but you can grab the full thing from **FileSilo** as well as the link below.

First we define a few basic parameters: the way the Hexy will walk and some of its speed parameters. We've also got a rotation parameter which we've set to 180, but you may need to tweak it for your Hexy. There's also the steps variable created just to make the code slightly easier.

Next we create a loop where for the first and last ten steps, the legs are articulated in order to make the Hexy move forward. This is a quarter of the walk forward section of the code, and once all parts have been completed we increase the step value by one. When it has reached ten steps, we do a load of code like in the last part to perform a full 180 degree turn, and then it does ten steps back with another if statement stopping the loop when a further 20 steps have been made.

Code listing

```
deg = 25
midFloor = 30
hipSwing = 25
pause = 0.5
rotate_deg = 180
rotate_step = 30
steps = 0
rotations = 0

...

While True:
   if steps != 10:
      # replant tripod2 forward while tripod1
      # move behind
      # relpant tripod 2 forward
      hexy.LF.replantFoot(deg-hipSwing,stepTime=0.5)
      hexy.RM.replantFoot(hipSwing,stepTime=0.5)
      hexy.LB.replantFoot(-deg-hipSwing,stepTime=0.5)

...

else:
      # set neck to where body is turning
      hexy.neck.set(rotate_step)
      # re-plant tripod1 deg degrees forward
      for leg in hexy.tripod1:
          leg.replantFoot(rotate_step,stepTime=0.2)
      time.sleep(0.5)
      # raise tripod2 feet in place as tripod1
      # rotate and neck
      for leg in hexy.tripod2:
           leg.setFootY(int(floor/2.0))
      time.sleep(0.3)
```

Next step

The above step-and-turn code can easily be adapted into a full-on catwalk routine, with tilts, leans, belly flops, audience-pointing and even a dance routine, should you wish to go all out. Just grab the PoMoCo source code (bit.ly/1ykuLQF) and work those Python chunks into your main script wherever you like.

Get the code

bit.ly/129HXMK

Above There are three parts to each leg and each part contains one servo. This articulation could potentially enable Hexy to climb over medium-sized objects and obstacles

Frindo

The puck robot with a low profile and plenty of front-facing sensors

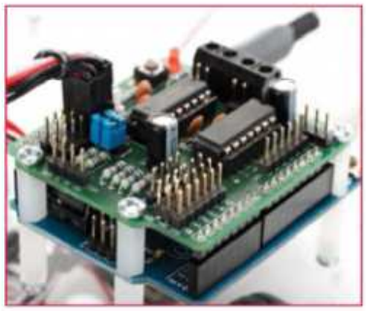
The Frindo is sold more as a robotics platform than an actual all-inclusive robot on its own, but that doesn't mean it's a very basic base to be built upon. Out of the box you can do a fair bit with the Frindo, while it's still extremely easy to build upon thanks to its support of standard Arduino and Raspberry Pi boards.

Construction is a very straightforward and quick process, although you will have to solder on wires to the motor during the construction. This is the only soldering that needs to be done on the Frindo though and it's very basic stuff. However, it is an extra step on top of everything else that not everyone may be equipped for. Still, the actual chassis construction and fitting of the motors and boards and wheels is done with very few components and can be completed quite quickly.

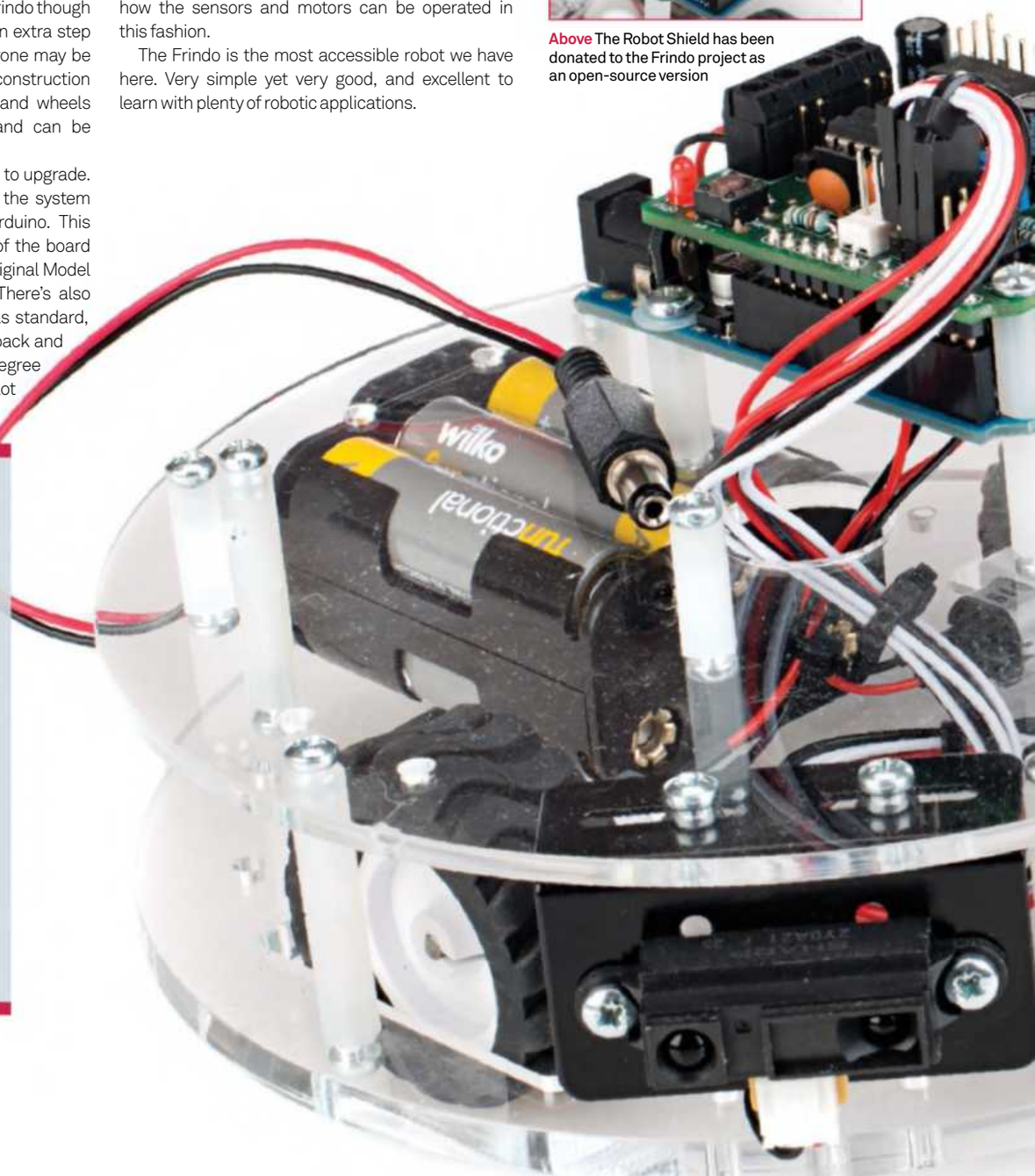
Once it's done you have a few options to upgrade. Firstly, you can add a Raspberry Pi to the system either with or without the supplied Arduino. This can be mounted on the opposite side of the board using holes specifically cut out for the original Model B (though unfortunately not the B+). There's also room for four more proximity sensors as standard, attachable in the spaces between the back and front sensors to create complete 360 degree coverage. The Uno and Pi can take a lot more inputs and outputs as well, so adding custom components is pretty easy.

Due to the dual controller support, the Frindo can be programmed in both Python and the Arduino IDE. Arduino uses the standard libraries and commands, making it great for those already up-to-speed with Arduino programming. The Python program uses the serial library, which uses terminology similar to Arduino, and there's a good, basic example on the website that can help you understand exactly how the sensors and motors can be operated in this fashion.

The Frindo is the most accessible robot we have here. Very simple yet very good, and excellent to learn with plenty of robotic applications.



Above The Robot Shield has been donated to the Frindo project as an open-source version



Technical specs

Manufacturer
Frindo

Height
85 mm

Width and depth
160 mm diameter

Weight
0.55 kg

Power
9 volts from 6 AA batteries

Control board
Arduino and/or Raspberry Pi

Form of locomotion
Wheels

Sensors
Four infrared proximity sensors

Website
www.robotbits.com

CHALLENGE 4...

Proximity sensor

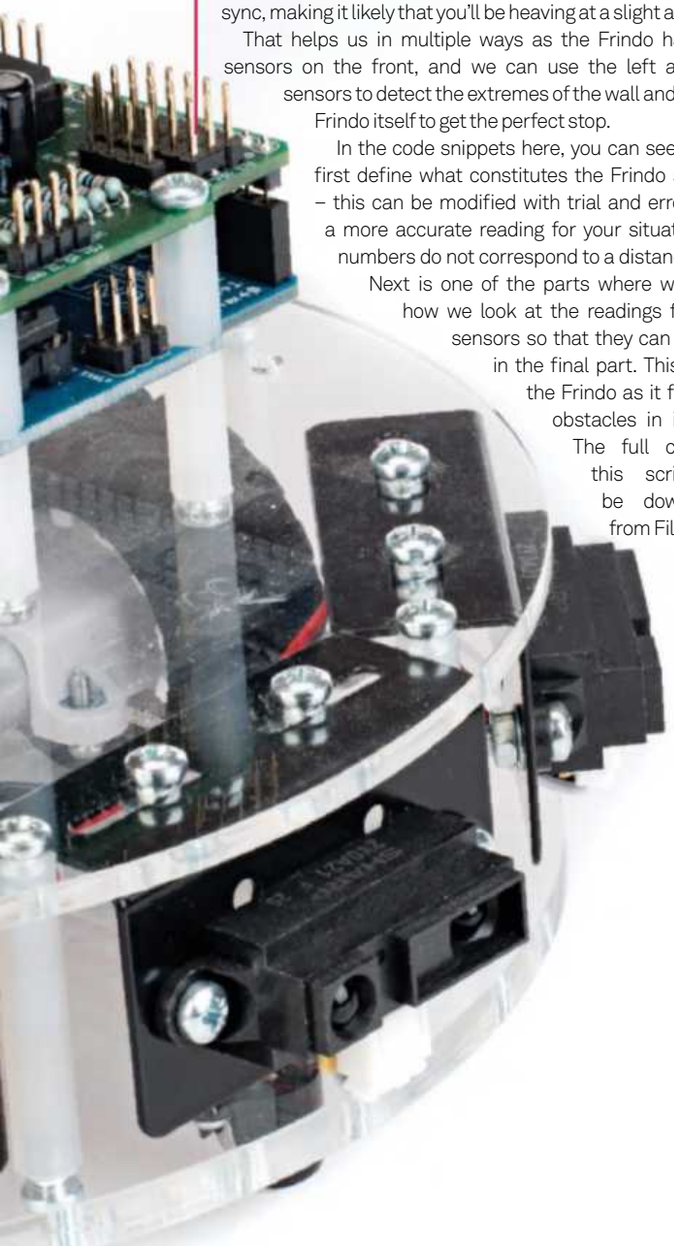
How close do you dare to go to the wall at the end of the course?

This challenge is somewhat simple: drive right up to a wooden wall and stop before hitting it. The closer you are before you stop, the more points you get. No touching of the wall is allowed. Should be easy with all those proximity sensors, right? Well it's not as easy as you would think, as the proximity sensor is not analogue. Surely there must be a way around it though?

The Frindo's sensors have some form of distance sensing on them, although it's by no means perfect. The other thing you'd have to calibrate for is program speed and stopping distance – and that's assuming you're heading straight on to begin with. The motors on the Frindo are unlikely to be in full sync, making it likely that you'll be heaving at a slight angle

That helps us in multiple ways as the Frindo has three sensors on the front, and we can use the left and right sensors to detect the extremes of the wall and turn the Frindo itself to get the perfect stop.

In the code snippets here, you can see that we first define what constitutes the Frindo stopping – this can be modified with trial and error to get a more accurate reading for your situation. The numbers do not correspond to a distance value. Next is one of the parts where we define how we look at the readings from the sensors so that they can be used in the final part. This rotates the Frindo as it finds any obstacles in its path. The full code for this script can be downloaded from FileSilo.



Code listing

Get the code
bit.ly/121Xa38

```
int frontTrigger = 200;
int sideTrigger = 100;
int rearTrigger = 100;

...

int front_bump() {
            bump = analogRead(FrontBump);
            if(bump > frontTrigger){
                return 1;
                }
            else {
                return 0;
                }
            }

...

void loop() {

            Serial.println("Here we go...");

            while(!front_bump()){
            // while there is no bump keep going forward
            // (about 10cm with GPD120)

                if(!left_bump() && !right_bump()) {
                    Serial.println("NO bump detected - move forward");
                    rs.forward(500, 200);
            // move forward for 500 mS at speed 200
            // (200/255ths of full speed)
                }

                else if(left_bump() && !right_bump())  {
                    Serial.println("LEFT bump detected - wrong angle");
                    rs.rot_ccw(100, 200);
            // turn right for 100 mS at speed 200
            // (200/255ths of full speed)
                }

                else if(!left_bump() && right_bump())  {
                    Serial.println("RIGHT bump detected - wrong angle");
                    rs.rot_cw(100, 200);
            // turn left for 100 mS at speed 200
            // (200/255ths of full speed)
                }
            }
```

The Frindo's sensors have some form of distance sensing on them

Rapiro

It stood up! The bipedal, humanoid, glowing-eyed, Arduino and Pi-powered robot straight out of Japan

Technical specs

Manufacturer
Kiluck

Height
257 mm

Width and depth
196 x 159 mm

Weight
1.00 kg

Power
7.5 volts from 5 AA rechargeable batteries

Control board
Custom Arduino (ATmega 328P) with optional Raspberry Pi

Form of locomotion
Bipedal walking

Sensors
Support for Pi camera

Website
www.rapiro.com

Programming the motors and servos are quite easy

The Rapiro is very unique on this list, even when compared to something like the Hexy. We were actually discussing in the office the difference in its design: Rapiro looks like a proper robot with its vacuum-formed shell, which in a way puts form over function. Not that it lacks function, but it's clear its creator Shota Ishiwatari fitted the motors around a design idea rather than design around the functions. It's a bit life-imitating-art, with Rapiro's design referencing robots in Japanese media compared to the hyperfunctional American and British robots with their ultrasonic sensors, line sensors and better stability that are more in line with some Hollywood films out there.

Construction of Rapiro is quite simple; you attach the myriad motors to different parts as you assemble the shell around them and thread the wires into his chest cavity where the Arduino lives. It's not really that fiddly, and there's no soldering or wiring involved. All the motors just plug into the board using the straightforward labelling you're asked to do in the manual early on.

While the assembly manual is not written by a native English speaker, the repetition and illustrations are generally easy enough to follow along to. Connecting a Raspberry Pi is not covered in the manual, but the Wiki shows where the connections between the Arduino and the Pi should be made, while the mount points are pretty obvious while constructing the head.

Programming the motors and servos are quite easy, with a number of preset serial commands enabling you to create custom scripts for the Rapiro to move or react a certain way to different inputs. This kind of autonomy can be achieved by using the Raspberry Pi and its camera to detect motion or specific objects, or respond to commands sent wirelessly to the board.

It's not the most sophisticated robot on this test, however there's nothing else that can properly walk on two legs either, or grip things. It's unique and useful for different tasks in comparison to the wheeled robots in our selection.

CHALLENGE 5...

Robot golf

It's a dog-leg par-four and Rapiro's taking a swing at it

It's actually more of a putting challenge, with the robot tasked to manoeuvre a small ball across a defined space into a goal. The goal is of mouse-hole design, meaning it just needs to be pushed in. While this challenge was envisioned with wheeled robots in mind, we decided we could take it a step further and have the Rapiro knock the ball into the hole with some well-placed swings of a tiny and light gold club. Time is the measure of success, so how would the Rapiro best complete the challenge?

While the Rapiro has superb articulation, it doesn't really have the ability to adopt a traditional golfer stance. Its arms can't cross and it doesn't really bend down. So what we plan to have it to do is hold a golf club and twist its body to hit the ball – very simple, yet effective. Not particularly accurate though, but one step at a time.

You'll see an excerpt of the Arduino script we're using to control the Rapiro, using the test script you can grab. It allows you to set eight movements for the Rapiro to make – this includes the angle of the 12 servos listed in a specific order, the three RGB values of the light and the time the action takes.

In our code, the Rapiro's eyes turn purple (with the mixture of 100 red and 150 blue) and it raises its arm quickly. We have two of the same pieces of code both taking '1' unit of time for this to occur. After that it opens its hand and changes colour to green, giving you time to put a 'golf club' in its hand. It then grips it, turning its eyes yellow to let you know it's getting ready to swing. Finally, it twists its waist to swing the club. The full code is available on FileSilo, although you may have to tweak it to work with your Rapiro's calibrations.

Next step

Next issue, we're going to show you how to use the Rapiro's head-mounted Raspberry Pi in order to remotely control him. But what then? Well the third eye can actually be replaced with a Ras Pi camera module, so with a little programming you would be able to add a cam feed to the control interface and see where you're going as you walk Rapiro around your home-made golf course. With Rapiro's built-in voice controls, you could also have a lot of fun approaching and interacting with your friends.

Left You can pull off some surprisingly delicate manoeuvres

Get the code
bit.ly/1HKBXeb

Code listing

```
{  // 10 Golf
  { 90, 90, 90,130, 90,180, 50, 90, 90, 90, 90, 90,100,  0,150,  1},
  { 90, 90, 90,130, 90,180, 50, 90, 90, 90, 90, 90,100,  0,150,  1},
  { 90, 90, 90,130, 90,180, 50, 90, 90, 90, 90, 90,  0,255,  0, 40},
  { 90, 90, 90,130,  0,180, 50, 90, 90, 90, 90, 90,255,255,  0, 10},
  { 90, 90, 90,130,  0,180, 50, 90, 90, 90, 90, 90,255,255,  0, 20},
  { 90,180, 90,130,  0,180, 50, 90, 90, 90, 90, 90,100,  0,150,  1},
  { 90,180, 90,130,  0,180, 50, 90, 90, 90, 90, 90,100,  0,150,  1},
  { 90,180, 90,130,  0,180, 50, 90, 90, 90, 90, 90,100,  0,150,100}
}
```

GoPiGo

The simple and straightforward Pi project robot with WASD control

Get the code
bit.ly/1ypCQod

Technical specs

Manufacturer
Dexter Industries

Height
85 mm

Width and depth
130 x 210 mm

Weight
0.60 kg

Power
12 volts from 8 AA batteries

Control board
Arduino (ATmega328) and Raspberry Pi

Form of locomotion
Two-wheel drive

Sensors
None – optional Pi camera optical sensor kit

Website
www.dexterindustries.com

GoPiGo is one of the simplest kits in the array we're testing here – simple in a good way though, with none of the negative connotations. The promised 20-minute build time is no exaggeration and we were up and running with this robot in no time at all. With no soldering required either then this really is an ideal gift for anyone interested in putting their first bot together. Given the sub-$100 (£63.80) price point it also makes an excellent base upon which to build more advanced projects, with plenty of room around the Pi and controller board within the open-sided shield for your own sensors and augmentations.

Straight out of the box, GoPiGo will work with Dexter Industries' firmware to give you WASD control of the two-wheel robot (the ball bearing caster at the rear making this a tricar of sorts), though nothing else beyond basic speed control. Being a Rasperry Pi-based project though, developing more advanced motion scripts and control for things like the optional ultrasonic sensor and camera module is a straightforward task.

There is one criticism to make, however: it seems there's a flaw with the design in that we found it impossible to connect the wheels properly. The wheels simply slip onto the end of the axles, and can very easily be popped off with a quick knock. The short axle length and nuts that graze the inner tyre wheels mean that it's difficult to actually push the wheels far enough onto the axles to give you the confidence that it'll hold together while driving. But that aside, and given the otherwise sterling quality of GoPiGo, we still feel that this is one of our favourite kits.

Sumo battle

Our 'gentlerobots of strength' tested their torque in the fine tradition of sumo wrestling. The rules were simple, though slightly different to those of the more well-known Homo sapiens variant of this popular robosport. Matches could not be won by forcing the opposing automaton to touch the ground with any part of their body other than the soles of their feet – largely because this would be impossible in most cases – but were instead focused on forcing them out of the dohyo (our tape-marked robot arena). It's a test of pure power, with each combatant driving forth and attempting to push back the other.

Scores explained

Here's a breakdown of our verdicts on these robots' qualities and capabilities

Rover5

Assembly 3/5
A little tricky in practise but still quite solid

Build quality 4/5
Generally fine but some of the screws are a little cheap

Programmability 3/5
For those without Arduino experience it can be a little confusing

Functionality 5/5
Great traction, great control and aware of its surroundings

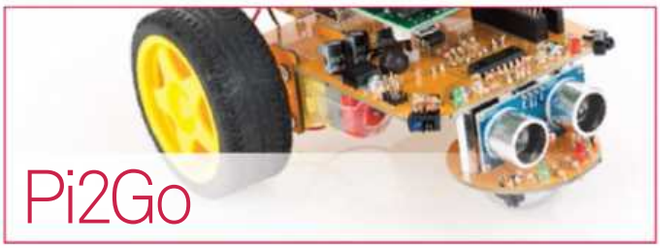
Rapiro

Assembly 4/5
Time-consuming but not fiddly due to its size

Build quality 5/5
It's very sturdy with a low centre of gravity

Programmability 3/5
Very simplistic Arduino commands are available

Functionality 2/5
Rapiro can move by your commands and that's about it



Pi2Go

Assembly 3/5
Soldering the kit together is time-consuming and not easy

Build quality 3/5
It's perfectly stable, but the chassis is its circuit boards

Programmability 4/5
A custom Python library makes it fairly easy to program

Functionality 5/5
For its size and price it has an absurd amount of features

Frindo

Assembly 4/5
Simple and quick; the basic chassis is easily constructed

Build quality 4/5
Very sturdy due to its shape and all components are protected

Programmability 4/5
If Arduino isn't your thing, you can always code it in Python

Functionality 4/5
The Frindo comes with three sensors but can be upgraded

Hexy the Hexapod

Assembly 3/5
Fairly straightforward, but the wide array of screws doesn't help

Build quality 4/5
Generally quite solid, but not consistent due to the screws

Programmability 5/5
PoMoCo gives full control over Hexy using visual aids and sliders

Functionality 5/5
Movement capabilities are incredible enough, but it does more

GoPiGo

Assembly 5/5
Simple and quick construction takes less than half an hour

Build quality 3/5
Generally okay, but the wheels have a problem staying on

Programmability 3/5
Can use simple terminal commands and be fully programmed

Functionality 1/5
GoPiGo can move on its wheels, but it has no sensors built-in

Remotely control your Ras Pi robot

Take Rapiro for a spin without it needing to be leashed to a laptop by controlling it over the network or with a PS3 controller

Rapiro is a cool little robot, and while we were playing around with it for our big robots guide we noticed that its potential for expansion and customisation was much simpler and satisfying than some of the other robots. In the spirit of that, we're going to slowly build up our Rapiro to do some amazing things, thanks to the ever-useful Raspberry Pi and a little Python know-how.

This month, we're going to liberate the Rapiro from the computer, and have it able to walk freely around on its own using the power of the Raspberry Pi and its own small selection of AA batteries to power it all. We're also going to teach you how to control it with a PS3 pad so that you don't have to constantly send commands via an interface.

What you'll need

- Raspberry Pi Model B
- Rapiro
- Compatible wireless dongle
- PS3 controller
- Mini USB cable

01 Set up the Raspberry Pi

You'll need an original Raspberry Pi Model B for the Rapiro, as the B+ needs you to trim the connections in the Rapiro for it to fit. Install Raspbian on it and perform the usual **rpi-update** to upgrade the firmware before doing a **sudo apt-get update && sudo apt-get upgrade** to update all the software.

02 Set up wireless

Keep Raspbian in command line-only boot (or set it to boot to command line in raspi-config), but stay on the desktop for the moment and use the graphical utility to set the wireless dongle up to connect your network.

03 Get the IP address

Open up the terminal and type in **ifconfig** – it will spit out the connection details for your networking. Specifically, you want to look for wlan0 and its relevant IP details. Make a note of this information, as you'll be using it to connect to the Pi remotely.

04 Connect remotely

Get on your PC or laptop and begin to test out the Raspberry Pi. Open a terminal and type in:

```
$ ssh pi@[IP address]
```

You may have to enable the keys on your machine, but enter the pi user password (raspberry) and you'll be into the Raspberry Pi's system.

05 Install extra software

To send the signals to the Rapiro from the command line, you need to get Minicom to send them. This is only required for the command line as Python has a special library for this. You can install this on the command line via SSH using:

```
$ sudo apt-get install
```

Above It's roomy enough above and below the Pi for a small add-on or two

06 Install into Rapiro

Turn off and disconnect the Raspberry Pi from all the cables and such. Next, remove the screws holding Rapiro's head together and take the front off. The Raspberry Pi delicately clips into the base of the head, with the ports poking out the back of the head – hold the back as it's not directly attached any more.

07 Connect the Raspberry Pi

The spare Pi cable left inside Rapiro's head can be connected to the Raspberry Pi to both power and allow for direct control from the Pi. Refer to the image to see how it's attached, with the black cable on the opposite side of the edge of the pins.

Above It takes a little practice but the analogue sticks are really fun to use

08 Reconnect the head

That's it! Once you've plugged the Raspberry Pi in, it's ready to go. Reattach the head to the body and you can start remotely controlling the Rapiro from the Pi. If you want to power Rapiro with external power at first, keep the head open as you'll need to also power the Pi at the same time.

09 First test

Turn the Rapiro on and see if everything works. Does the Raspberry Pi light up? Do the motors work fine? The most common problem is that you may have put the Raspberry Pi cable into the torso in the wrong way, meaning you'll have to completely disassemble it to fix it.

10 First commands

If your Rapiro is now working and the Raspberry Pi is powering on, it's time to take it for a test drive. Connect via SSH like before and get Rapiro to wave by calling the M6 command using the following command:

```
$ echo "#M6" | sudo minicom -b 57600 -o -D /dev/ttyAMA0
```

11 Serial commands

This is using the Arduino sketch that has been previously uploaded to your Rapiro, with M6 being the left wave. You can also directly control the motors and LEDs with serial commands. To change the lights to green over 10ms, enter:

```
$ echo "#PR000G255B000T010" | sudo minicom -b 57600 -o -D /dev/ttyAMA0
```

12 Break down the command

There are four parts to that command: the three RGB values for the LED lights and the T value for time in milliseconds. We've set the green to maximum (255) and the other two colours values to 0, while T is 10. Quite simple – play around with the code.

13 Raise right hand

Let's do something a little more interesting, and move one of the servos in Rapiro's body. We can have it raise its right arm by sending a couple of commands to its right arm servo like so:

```
$ echo "#PS02A000T010#PS02A180T010" | sudo minicom -b 57600 -o -D /dev/ttyAMA0
```

14 Further breakdowns

There's more in this serial code to explain. First of all, the S number indicates which servo should be used – in this case 02, which is the right shoulder servo. The A value is the angle of the motor: 000 is standard setting of the arm being down, while 180 is straight up. T is used the same as before in both commands. Only two can be used at a time like this.

15 Full code listing

From here you can start making custom scripts for Rapiro that do a lot more than some of the Arduino scripts. They can be a bit clunky, though, due to the way you'd have to execute them, but by referring to the table on the following page you'll be able to build up an appropriate script for any custom commands you want to do outside of Arduino.

To get Rapiro to work with the controller, you'll need a custom Python script

Serial Code Table

S00	The head motor, turn side to side
S01	The waist motor, turn side to side
S02	Right shoulder motor, rotating the arm forward and back
S03	Right arm motor, raising the arm up and down
S04	Right hand motor, open and close the hand
S05	Left shoulder motor, rotating the arm forward and back
S06	Left arm motor, raise the arm up and down
S07	Left hand motor, open and close the hand
S08	Right foot yaw motor
S09	Right foot pitch motor
S10	Left foot yaw motor
S11	Left foot pitch motor
R	Red colour for eye LEDs, 0 to 255
G	Green colour for eye LEDs, 0 to 255
B	Blue colour for eye LEDs, 0 to 255
T	Time in milliseconds for the action to take place

16 Control code

To get Rapiro to work with the PS3 controller, you'll need a custom Python script. A basic script can be found on the Rapiro website, so download it to the Pi in your Rapiro with:

```
$ wget http://www.kiluck.co.jp/rapiro/rapiro_ps3_ver0_1.py
```

17 Preparation

Grab a PS3 controller and the charge cable, or any available USB mini cable you might have. Plug it into the USB port and wait a few seconds for it to connect to the Pi. You can then run the code using:

```
$ sudo python rapiro_ps3_ver0_1.py
```

18 Move Rapiro!

Okay, this is the cool part: the analogue sticks both control the arms, each being able to manipulate the rotation of the shoulder and raising the arm. Pulling the triggers will grip one of the hands, while the normal L and R buttons swivel the head. The D-pad buttons allow you to activate the walk forward and back commands as well as turn on the spot. The face buttons allow you to wave with each hand or both hands, with the X button stopping everything and putting it back to the base state.

19 Code limitations

The code is limited to how many actions you can perform at once, and will overload the serial buffer if you try and do anything too complicated. That's fine, though, as you can immediately start the code again.

20 Modifying the code

The code calls for specific moves from the standard Rapiro sketch for the face buttons. Using the code for Rapiro last issue where we added the golf ability, we can also add this action to Select. Find the line com.write in the SELECT part of the code and change it to:

```
com.write("#M9")
```

Alternatively, grab this final code from File Silo.

Truly wireless

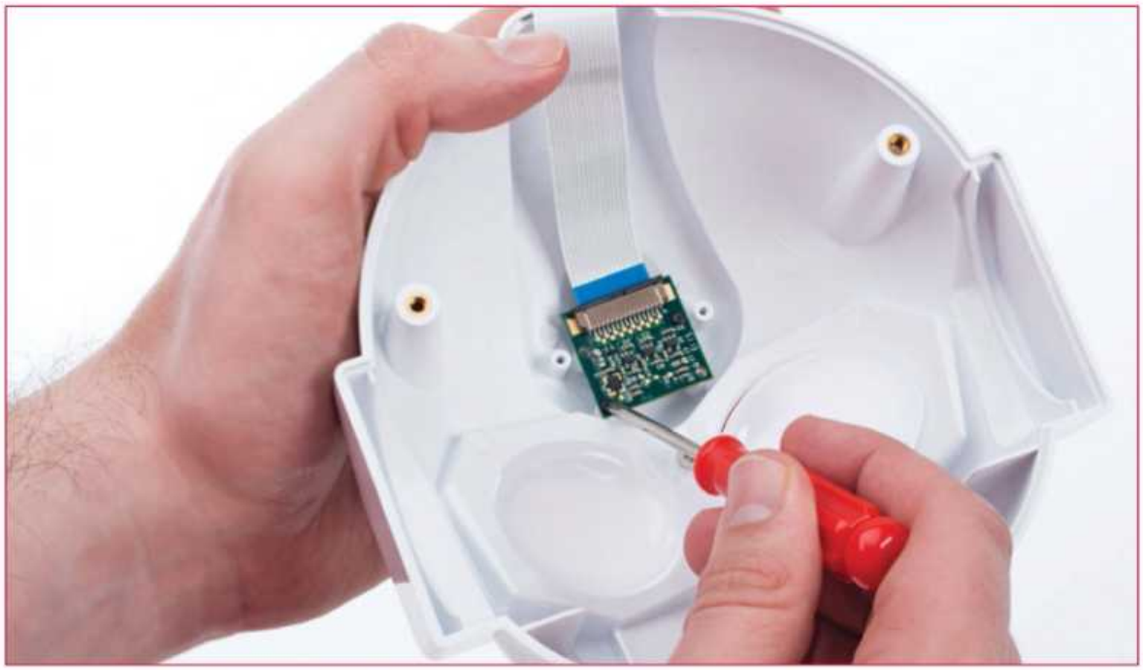
The PS3 controller is connected to the Rapiro with a wire like a leash, reminding it who its master is, however with a bit of hackery you can install a Bluetooth USB module and modify the code so the controller connects wirelessly. Give it a shot as an upgrade!

Add web control and a camera to Rapiro

Make the Rapiro more wireless by installing a remote web interface and see where it's going in the process

What you'll need

- Raspberry Pi Model B
- Raspberry Pi Camera
- Latest Raspbian image
  raspberrypi.org/downloads
- A wireless dongle



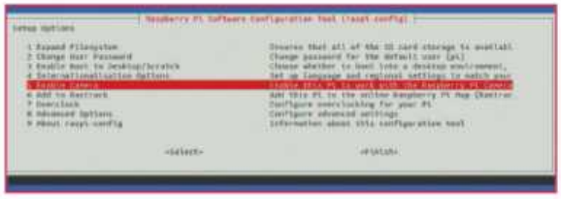
Left The design of the Rapiro means that it's quite simple to figure out where everything should sit

So, we have just added the ability to control the Rapiro using a PS3 controller. Now we want to give even better remote control to the user, so this time we're adding a web interface and installing the Pi camera into the head of the Rapiro so you can see what it's looking at.

Theoretically, as long as there's a constant wireless signal, you can see and direct the Rapiro as you wish. The only real limit is battery power. Part of this tutorial will show you how to set up a web server on the Raspberry Pi so that it can handle all the robotics and commands, and also stream the camera from the Rapiro.

01 Set up Raspbian

First, let's make sure we get Raspbian properly set up. Update your SD card or write the latest version to the card itself. For the update do the usual **sudo apt-get update && sudo apt-get upgrade**. If you've written a new card, insert it now.



02 Further set up

With a new card, turn on the Pi and get to the configuration screen. On an existing Raspbian, use **sudo raspi-config** in the terminal. Set the desktop to CLI to conserve the power, extend the file system and enable the Raspberry Pi camera module.

03 Connect to the Wi-Fi

Give it a quick reboot once you've finished configuring. If you haven't set up the Wi-Fi yet, the easiest way to do so is to go into the desktop with **startx** and use the graphical wireless network tool to set it up. It will save these settings even when booting into the command line. Make sure to also get the IP address assigned to the Pi using **ifconfig**.

04 Install the Raspberry Pi

Now that everything is set up, turn the Pi off and open up your Rapiro. Put the Pi in its slot and connect up the Arduino cable from the base. Take the front piece of the head and remove the plastic part above the eyes that plugs into the camera hole.

05 Attach the camera

Use the same screws that attached the plug to install the Pi camera – do it upside down so that the cable can fit. Once it's affixed securely, attach it to the Raspberry Pi in the camera port with the silver of the cable facing towards the front of the Rapiro. Reattach the head to the body.

06 Log in via ssh

Now your Rapiro is reassembled, plug in your Wi-Fi adapter and turn the Rapiro on. Give it a few moments and then try and log into the Raspberry Pi via ssh. From the terminal on another computer, use:

```
$ ssh pi@[IP address]
```

07 Disable getty

We need to disable the normal login screen on the Raspberry Pi so we can use the serial port. Access the init file using **sudo nano /etc/inittab** and find the following line:

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

Comment it out by adding a # in front of the line.

3D-print Rapiro

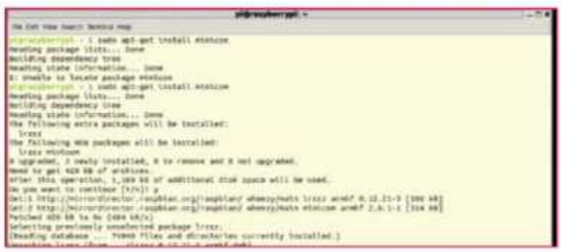
As promised by creator Shota Ishiwatari during its initial Kickstarter campaign, the 3D models for Rapiro are now available to download for free from Thingiverse, so if you or a friend have a 3D printer, or you're happy paying a visit to one of the many 3D printing shops now appearing on the high streets, you can 3D-print your very own Rapiro. These days you can even print in colour, as well as all apply all sorts of finishing effects like metal and wood. Download the files from: **thingiverse.com/thing:309466**.

Above Be careful not to knock the camera module's ribbon out of place when you reattach the face

08 Stop serial ports at boot

Once the inittab is saved, we need to stop the Pi from trying to send out data to the serial port when it boots. Use **sudo nano /boot/cmdline.txt** and delete the following from the file before saving and rebooting the Pi:

```
console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
```



09 Install command software

Using the Rapiro in the Pi means that you need extra software – if you did the remote control tutorial last time, you probably already have it. Just to be safe though, install it with:

```
$ sudo apt-get install minicom python-serial
```

10 Test the ports

Before we start testing what we've done, you'll need to change some permissions to a most foul number. Once it's done though, we can write a little Python script for testing:

```
$ sudo chmod 666 /dev/ttyAMA0
```

11 Do a script test

Create a new Python file in the terminal with **nano remotetest.py** and enter the following:

```
import serial
import time
ser = serial.Serial('/dev/ttyAMA0', 57600, timeout=1)
ser.open()

ser.write("#PR255G000B000T050")
time.sleep(2)
ser.write("#PR000G000B255T050")
time.sleep(2)
ser.write("#H")

ser.close()
```

12 Run the test

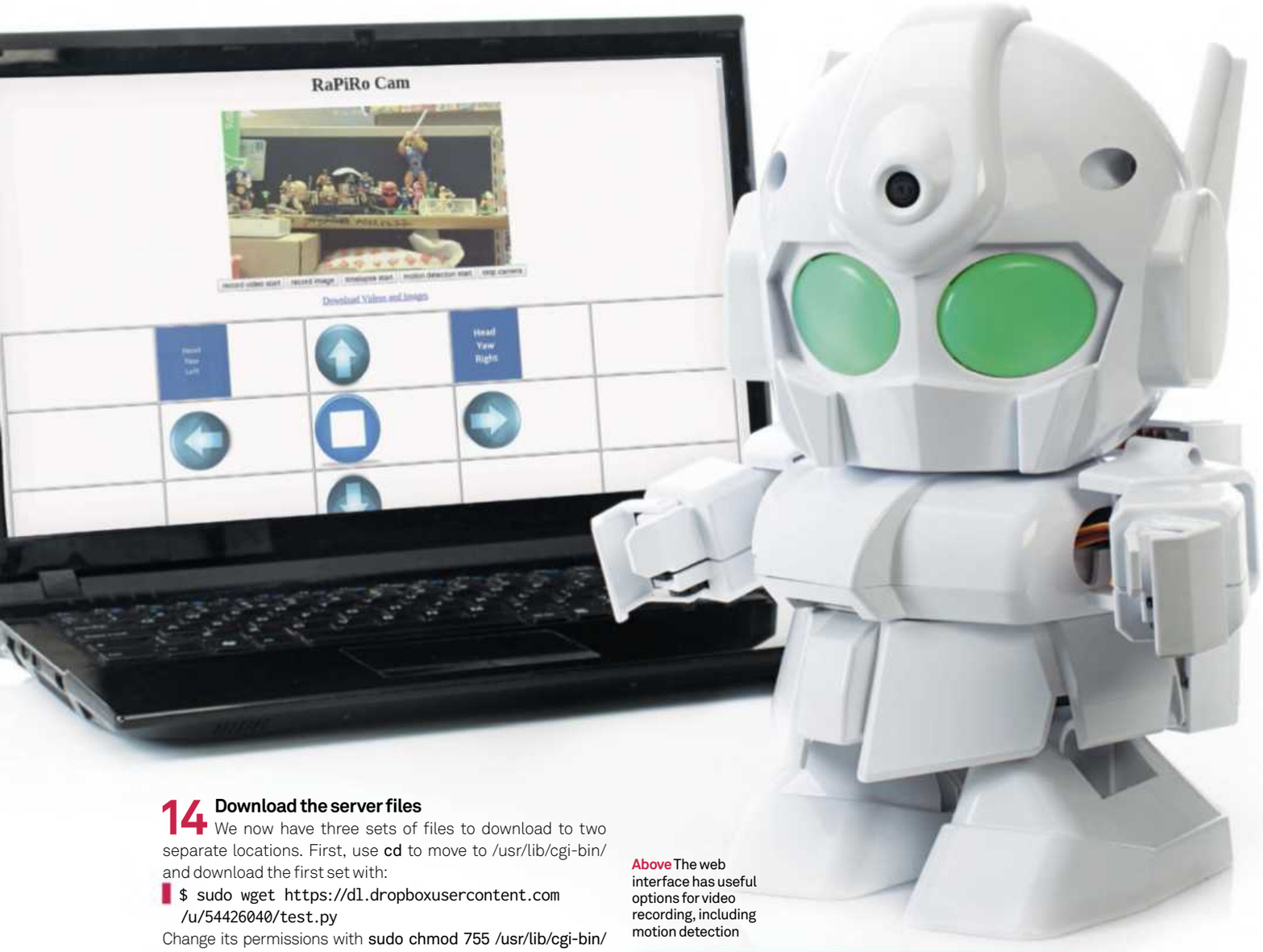
Save the file. This should change the eye colour on the Rapiro to red then back to blue. Run it and any other custom python scripts for Rapiro by using:

```
$ python remotetest.py
```

13 Install Apache

To get the web sever working, we now need to install Apache onto the Pi. Do that and then configure it with:

```
$ sudo apt-get install apache2 libapache2-mod-python
$ sudo usermod -aG dialout www-data
```



Above The web interface has useful options for video recording, including motion detection

14 Download the server files

We now have three sets of files to download to two separate locations. First, use **cd** to move to /usr/lib/cgi-bin/ and download the first set with:

```
$ sudo wget https://dl.dropboxusercontent.com
/u/54426040/test.py
```

Change its permissions with **sudo chmod 755 /usr/lib/cgi-bin/test.py** then **cd** to /var/www/ to download files two and three:

```
$ sudo wget https://dl.dropboxusercontent.com
/u/54426040/control.css
$ sudo wget https://dl.dropboxusercontent.com
/u/54426040/Remote.Control.Icons.zip
```

Unzip the last file with **sudo unzip Remote.Control.Icons.zip**.

15 Launch the interface

Finally, you can launch and control your Rapiro from **http://[IP address]/cgi-bin/test.py**. It's quite a basic interface that you can easily customise yourself to make it a little neater. This interface was created by eLinux user JaixBly.

16 Download the camera software

There is great camera software from Silvan Melchior on the Raspberry Pi forums that we can use to stream the Pi camera. Back in the terminal, download the files with:

```
$ git clone https://github.com/silvanmelchior/
RPi_Cam_Web_Interface.git
```

Use **cd** to move to Rpi_Cam_Web_Interface and make the installer executable with:

```
$ chmod u+x RPi_Cam_Web_Interface_Installer.sh
```

17 Install the camera software

With the software downloaded and set up, it's time to properly install it. Do this using the following command and then reboot the Pi/Rapiro:

```
$ ./RPi_Cam_Web_Interface_Installer.sh install
```

18 Test out the camera

After the reboot, the red LED on the camera should light up to indicate it's on. To see the stream, go to **172.25.12.84** in the browser to see a stream with lots of details. It may be upside down so rotate the image.

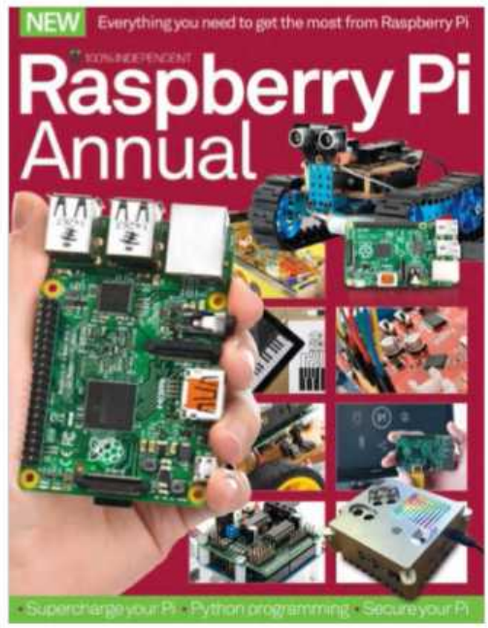
19 Add files to controls

Back in the terminal, download some extra files for the web interface with:

```
$ wget https://www.dropbox.com/s/jt6sn1o3rytau7t/
RaPiRo.zip
```

Unzip it with **unzip RaPiRo.zip** and copy the replacement files into their respective spots, do a **sudo chmod 755 /usr/lib/cgi-bin/rapiro.py** and then reboot.

Special trial offer



Enjoyed this book?

Exclusive offer for new

Try 3 issues for just £5*

* This offer entitles new UK Direct Debit subscribers to receive their first three issues for £5. After these issues, subscribers will then pay £25.15 every six issues. Subscribers can cancel this subscription at any time. New subscriptions will start from the next available issue. Offer code 'ZGGZINE' must be quoted to receive this special subscriptions price. Direct Debit guarantee available on request. This offer will expire 30 November 2016.

** This is a US subscription offer. The USA issue rate is based on an annual subscription price of £65 for 13 issues, which is equivalent to $102 at the time of writing compared with the newsstand price of $16.99 for 13 issues, being $220.87. Your subscription will start from the next available issue. This offer expires 30 November 2016

About the mag

The magazine for the GNU generation

Written for you

Linux User is the only magazine dedicated to advanced users, developers & IT professionals

In-depth guides & features

Written by grass-roots developers & industry experts

Jam-packed with Ras Pi

A Practical Raspberry Pi section brings you clever tutorials, inspirational interviews and more

subscribers to...

LinuxUser & Developer™

Try three issues for £5 in the UK*
or just $7.85 per issue in the USA**
(saving 54% off the newsstand price)

For amazing offers please visit
www.imaginesubs.co.uk/lud
Quote code ZGGZINE

Or telephone UK 0844 249 0282+ overseas +44 (0) 1795 418 661

+ Calls will cost 7p per minute plus your telephone company's access charge

YOUR FREE RESOURCES

Log in to **filesilo.co.uk/bks-819/** and download your tutorial resources **NOW!**

EVERYTHING YOU NEED TO FOLLOW THE TUTORIALS AND PROJECTS IN THIS BOOK

ubuntu®

GET THE SOFTWARE USED IN THE TUTORIALS

Python is the ultimate Raspberry Pi coding language

Complete the tutorials

redhat. LINUX

Download distros

PACKED WITH FREE PREMIUM CONTENT

YOUR BONUS RESOURCES

ON FILESILO WE'VE PROVIDED FREE, EXCLUSIVE CONTENT FOR **RASPBERRY PI ANNUAL VOLUME 2** READERS, INCLUDING...

- 31 distros to use in your projects including Ubuntu, BackBox 4.3, Raspbian, Peppermint OS 6, Fedora Security, OpenSUSE and more
- 14 pieces of software to download to help you complete your Raspberry Pi-based ideas including code and robot scripts
- 6+ hours of video tutorials across 20 videos covering everything from how to write good Raspberry Pi code and an insight into Debian Linux

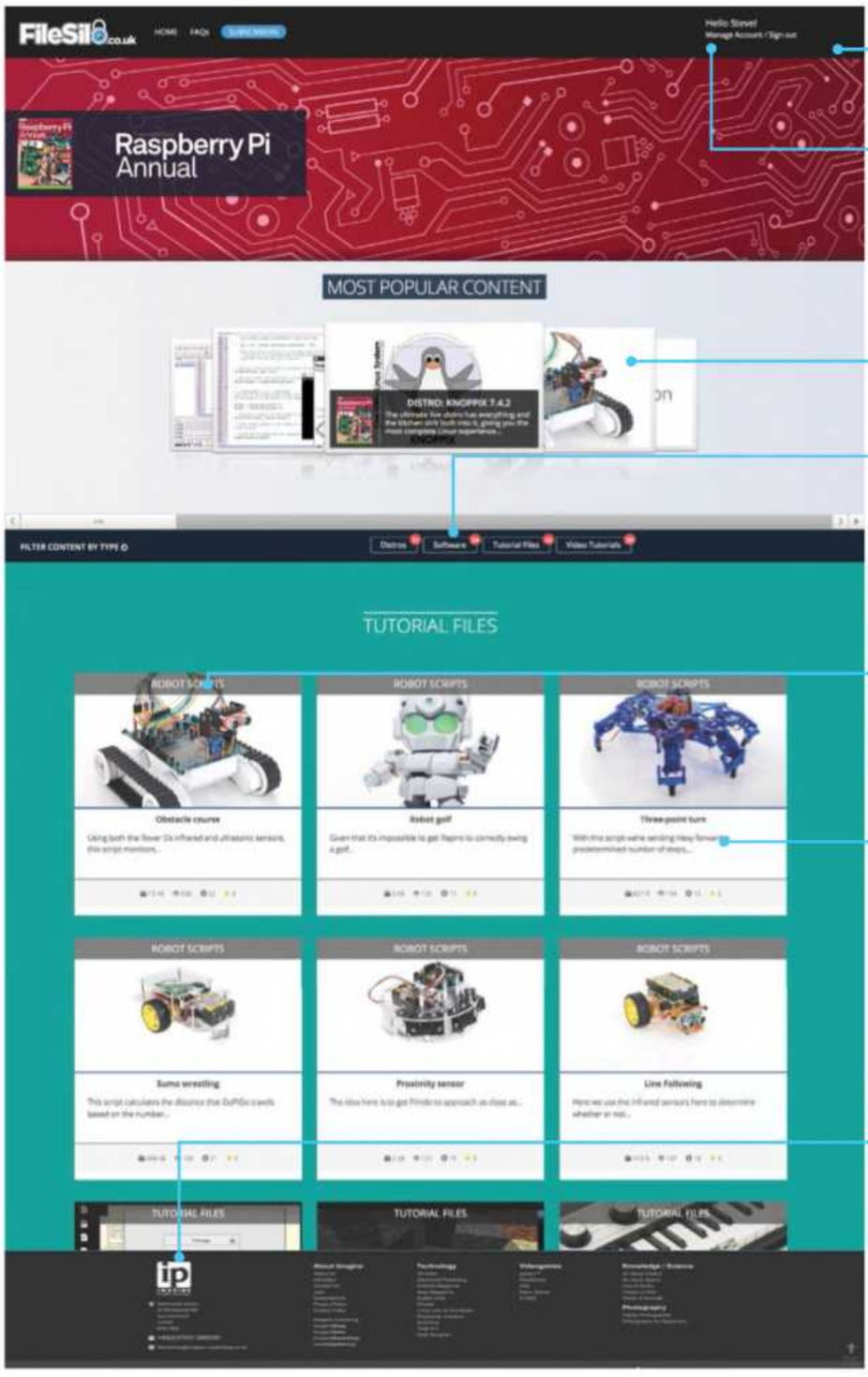
FileSilo

Go to: http://www.filesilo.co.uk/bks-819/

FILESILO – THE HOME OF PRO RESOURCES

Discover your free online assets

- A rapidly growing library
- Updated continually with cool resources
- Lets you keep your downloads organised
- Browse and access your content from anywhere
- No more torn disc pages to ruin your magazines
- No more broken discs
- Print subscribers get all the content
- Digital magazine owners get all the content too!
- Each issue's content is free with your magazine
- Secure online access to your free resources



This is the new FileSilo site that replaces your disc. You'll find it by visiting the link on the following page.

The first time you use FileSilo, you'll need to register. After that, you can use your email address and password to log in.

The most popular downloads are shown in the carousel here, so check out what your fellow readers are enjoying.

If you're looking for a particular type of content, like software or video tutorials, use the filters here to refine your search

Whether it's Python tutorials or software resources, categories make it easy to identify the content you're looking for

See key details for each resource including number of views and downloads, and the community rating

Find out more about our online stores, and useful FAQs, such as our cookie and privacy policies and contact details.

Discover our fantastic sister magazines and the wealth of content and information that they provide.

HOW TO USE FileSilo

EVERYTHING YOU NEED TO KNOW ABOUT ACCESSING YOUR NEW DIGITAL REPOSITORY

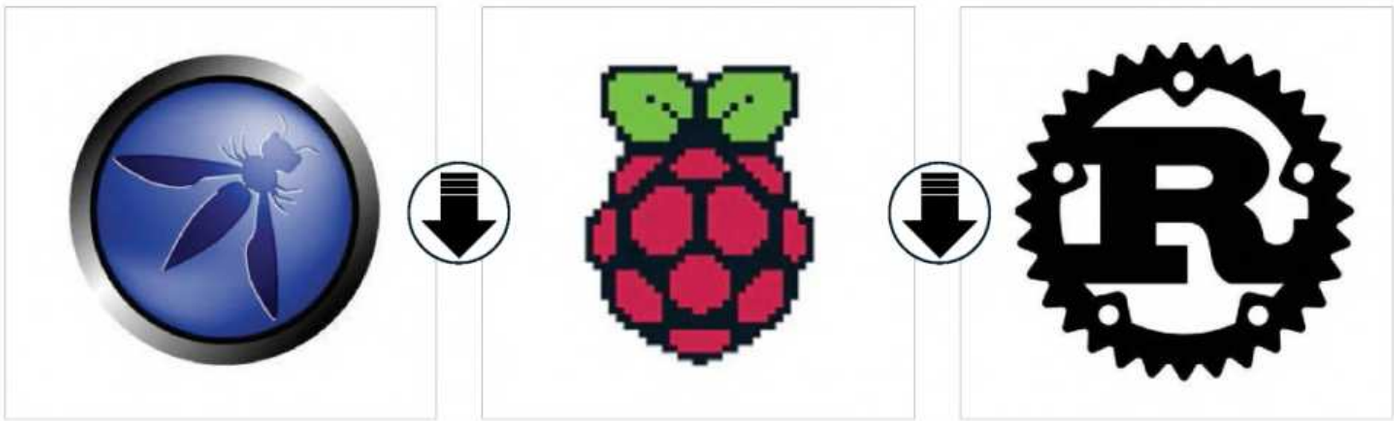
To access FileSilo, please visit http://www.filesilo.co.uk/bks-819/

01 Follow the on-screen instructions to create an account with our secure FileSilo system, or log in and unlock the issue by answering a simple question about the edition you've just read. You can access the content for free with each edition released.

02 Once you have logged in, you are free to explore the wealth of content made available for free on FileSilo, from great video tutorials and online guides to superb downloadable resources. And the more bookazines you purchase, the more your instantly accessible collection of digital content will grow.

03 You can access FileSilo on any desktop, tablet or smartphone device using any popular browser (such as Safari, Firefox or Google Chrome). However, we recommend that you use a desktop to download content, as you may not be able to download files to your phone or tablet.

04 If you have any problems with accessing content on FileSilo, or with the registration process, take a look at the FAQs online or email filesilohelp@imagine-publishing.co.uk.



NEED HELP WITH THE TUTORIALS?

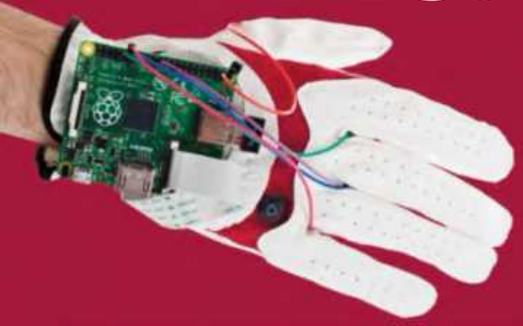
Having trouble with any of the techniques in this issue's tutorials? Don't know how to make the best use of your free resources? Want to have your work critiqued by those in the know? Then why not visit the Bookazines or Linux User & Developer Facebook page for all your questions, concerns and qualms. There is a friendly community of experts to help you out, as well as regular posts and updates from the bookazine team. Like us today and start chatting!

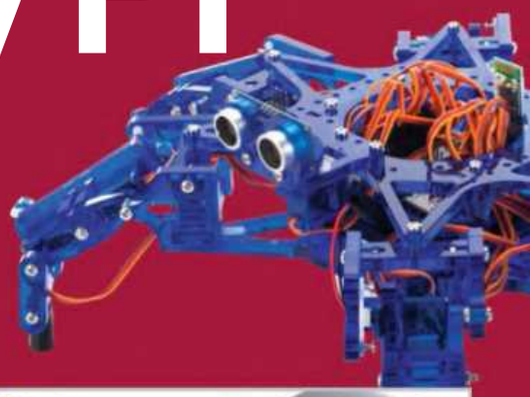
facebook.com/ImagineBookazines
facebook.com/LinuxUserUK

100% INDEPENDENT

Raspberry Pi Annual

Everything you need to get the most from Raspberry Pi





Practical projects

Discover the greatest Raspberry Pi hardware hacks to ever exist

Discover cutting-edge tech

Get the inside story on the revolutionary technology emerging around the device

Upgrade with accessories

Add functionality with the best add-on modules, cases and much more

Perfect your coding

Learn to speak Python, the world's most popular programming language for Pi

Build smart machines

Create your own innovative gadgets by combining your Pi with household tech

Take the next step

From artificial intelligence to a Minecraft console, make something truly inspiring